MCNelectron

Open-source Monte Carlo code for simulation of coupled electron-photon transport

v1.1.2

User‘s Manual

by Andrius Poškus

(Vilnius University, Department of Solid State Electronics)

2015-12-03

Copyright © 2015 by Andrius Poškus

E-mail: andrius.poskus@ff.vu.lt

Web: http://web.vu.lt/ff/a.poskus/mcnelectron/

Contents

1. Introduction 1
1.1. Comparison with MCNP6 1
1.2. Single-event method vs. condensed-history approximations 2
1.3. CUDA support in MCNelectron 3
2. Cross section data files 4
3. Compiling MCNelectron 5
4. Running Monte Carlo simulations with MCNelectron 6
4.1. Input file format 8
4.2. Format of the file with alternative cross section information 15
4.3. Plane-crossing tallies 17
4.4. Specifying the maximum error for plane-crossing and pulse-height tallies 19
4.5. Interaction forcing 20
4.6. CUDA-specific keywords 23
4.7. CUDA device statistics 28
5. Simulation of physical effects that are not included in specification of the ENDF/B library 32
6. MCNelectron output file format 37
7. Information about MCNelectron test files 38
8. Information about files in the subfolder “Simulations\X-rays” of the distribution package 40
References 41

1. Introduction

MCNelectron is an open-source single-event Monte Carlo code for simulation of coupled electron-photon transport. MCNelectron was written by Andrius Poškus (Vilnius University, Faculty of Physics, Department of Solid State Electronics). MCNelectron uses a public-domain Fibonacci series random number generator by George Marsaglia and Arif Zaman (downloaded from www.netlib.org). MCNelectron is distributed under the GNU General Public License.

MCNelectron can be downloaded from http://web.vu.lt/ff/a.poskus/mcnelectron/.

1.1. Comparison with MCNP6

The physics models implemented in MCNelectron are essentially the same as those applied by MCNP6 [1]. However, MCNelectron allows only the simplest possible geometry of the simulated system: a parallel beam of electrons, positrons or photons with arbitrary energy spectrum incident at any angle on an infinite homogeneous layer of arbitrary thickness and arbitrary composition. The beam radius is zero. There are no restrictions on position of the starting point of the beam (for example, it may be inside the layer). Tallying capability of MCNelectron is much more limited than that of MCNP. MCNelectron has only two types of tallies: a plane-crossing tally and a pulse-height tally (their MCNP counterparts are "F1" and "F8" tallies, respectively). All tallies have equidistant bins. In addition to the tallies, MCNelectron outputs the following statistics:

- total numbers of various types of energy-loss interaction events,
- total numbers of secondary electrons and photons created in various types of interactions,
- total energies of secondary electrons and photons created in various types of interactions,
- total energy losses of electrons and photons due to various types of interactions,
- the average energy loss per one secondary electron.

Only publicly-available information about physics models and simulation methods implemented in MCNP was used when writing MCNelectron. The author of MCNelectron has never had access to source code of any version of MCNP or any other Monte Carlo simulation software. MCNelectron has been written in order to circumvent some of the limitations of MCNP6:

- Simulation of low-energy (less than 1 keV) electron and photon transport in MCNP6 is done using only ENDF/B-VI.8 data (MCNelectron allows replacing ENDF/B cross sections of some types of electron interactions by alternative cross sections from other libraries);
- MCNP6 does CPU-only computations, whereas MCNelectron can perform so-called "hybrid" computations, when both the central processing unit (CPU) and Nvidia graphics processing units (GPU) are employed in parallel. This is achieved using CUDA (see Section 1.3 for details);
- MCNP6 does not allow setting the low-energy cutoff limit for electrons to values less than 10 eV (MCNelectron allows that);
- MCNP6 outputs only the statistics pertaining to *tracked* electrons and photons. Hence, if initial energy of a knock-on electron is less than the cutoff energy, that electron will not be counted in the total number of knock-on electrons that is reported in the MCNP output file. On the other hand, MCNelectron counts the number of ionization *events*, regardless of the knock-on electron energy. This allows to take into account the knock-on electrons with energies less than the cutoff energy;
- MCNP6 does not allow turning off elastic scattering of electrons (MCNelectron allows that);
- MCNP6 does not allow turning off tracking of particle coordinates (MCNelectron allows that).

The latter two options are useful for speeding up the simulation when calculating the average energy loss per one secondary electron due to complete absorption of incident particle energy in an infinite medium (in such a case, coherent scattering of photons should be turned off, too). However, even in the case of the maximum complexity of simulation, i.e., when all possible types of interactions are "turned on" and particle coordinates are tracked, a CPU-only simulation done with

MCNelectron is shorter than a simulation done with MCNP6 in single-event mode (under identical simulation conditions, the time of a CPU-only simulation with MCNelectron v1.1.2 is usually from 40 % to 70 % of the simulation time with MCNP6 in single-event mode, depending on the simulated system).

By default, the data used by MCNelectron are taken from the ENDF/B-VI.8 library, except for cross sections of inner-shell electroionization (they were calculated according to the distorted-wave Born approximation), cross sections of electron elastic scattering and corresponding angular distributions (they were obtained from relativistic (Dirac) partial-wave calculations), and cross sections of positron bremsstrahlung (they reflect the effect of partial suppression of positron bremsstrahlung in comparison with electron bremsstrahlung, which is not taken into account by the ENDF/B data). Those replacements of ENDF/B data are "on" by default, because they increase accuracy of the simulation results. However, the user has an option to revert to the ENDF/B data when simulating each of the mentioned interactions. In addition, it is possible to replace ENDF/B low-energy electroionization and excitation cross sections by third-party cross sections provided by the user (with optional interpolation between those low-energy alternative cross sections and high-energy ENDF/B cross sections). The data that are always taken from the ENDF/B library (i.e., can not be replaced) include electron bremsstrahlung cross sections, excitation energy transfer, all photon interaction cross sections and atomic relaxation data.

1.2. Single-event method vs. condensed-history approximations

In most available general-purpose Monte Carlo codes, electron transport is simulated using a combination of two methods – single-event (SE) and condensed-history (CH), with a possibility to choose between the two methods via a user-adjustable parameter. The SE method simulates individual electron collisions separately, whereas the CH approximation is based on breaking the electron's path into many steps, chosen long enough to encompass many collisions, which are treated by multiple-scattering theories. The need for the CH approximation arises from the fact that the average energy loss in one ionizing collision of an electron with energy much higher than the ionization threshold is of the order of 10 eV or less in most materials, i.e., relatively small. Consequently, as it follows from the law of conservation of momentum, the average angular deflection of the incident electron is also relatively small. Hence an electron with energy of the order of 10 keV experiences several thousand inelastic collisions with small-angle deflections, and an even greater number of elastic collisions before coming to a halt. As electron energy is increased, the number of collisions increases proportionally. This large number of collisions can make the detailed event-by-event simulation prohibitively long when simulating interaction of high-energy electrons with thick targets. The CH approximations make it possible to shorten the simulation significantly while preserving the essential physical features of the simulated process.

In comparison with the condensed-history approximations, the single-event method is conceptually much simpler. In the case of the SE method, motion of a particle is simulated by repeating those four steps:

1) calculate the total cross section and use it to calculate distance to collision,
2) select target (if there are several types of atoms in the material),
3) select the type of interaction,
4) calculate energies and directions of the primary and secondary particles after the interaction.

This scheme is always applied to photons, because photon transport is faster than electron transport, i.e., the number of photon collisions is typically much less than the number of electron collisions.

Since the SE method is less dependent on implementation-specific theoretical assumptions than the CH approximation, the SE method is potentially more accurate. Its accuracy is mainly determined by accuracy of cross sections used for the simulation.

MCNelectron simulates coupled electron-photon transport using only the single-event method (both for electrons and for photons), except at extremely low electron energies, when a diffusion model of elastic scattering may be optionally applied.

1.3. CUDA support in MCNelectron

There are two versions of MCNelectron: the CPU-only version (the executable file name MCNelectron.exe) and the version with CUDA support (MCNelectron_CUDA.exe). The file MCNelectron_CUDA.exe can perform so-called "hybrid" computations, when both the central processing unit (CPU) and Nvidia graphics processing units (GPU) are employed in parallel. The CUDA version of MCNelectron can also perform GPU-only or CPU-only computations (in CPU-only mode, MCNelectron_CUDA.exe produces the same results as MCNelectron.exe).

CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming model created by Nvidia Corporation [2]. It allows using Nvidia video cards for general-purpose processing. The GPUs used for the simulation need not to be dedicated for general-purpose computations: the video card that outputs images to a computer monitor may be used at the same time for the simulation. MCNelectron implements automatic workload balancing between the CPU and available CUDA devices during the simulation, so that the CPU and the GPU (or several GPUs, if there are two or more Nvidia video cards in the system) spend approximately equal times for the simulation. The GPUs must have CUDA compute capability 2.0 or higher. The improvement of performance when GPUs are used together with the CPU in comparison with a CPU-only computation depends on a particular CPU/GPU combination present in the system. For example, simulation time on a single Nvidia GeForce GTX 780 Ti video card is about 1/4 of the time required for the same simulation using a single thread on an Intel Core i7-4930K processor. However, from a practical point of view it is more meaningful to compare the simulation times when the CPU is at full load, i.e., when all cores of the CPU are used in parallel, because in practice one is interested in utilizing all available computing power as fully as possible. Since the Intel Core i7-4930K processor has 6 cores and uses hyper-threading, the maximum CPU load is achieved with 12 or more threads, and then the CPU is approximately 8 times faster than a single CPU thread (performance does not scale linearly with the number of CPU threads). Hence, an improvement in processing speed in hybrid computations is about 50 % when a single GTX 780 Ti video card is used together with the CPU, and about 100 % when two GTX 780 Ti video cards are used. This is evident in Table 1, which compares simulation times using an Intel Core i7-4930K processor and two Nvidia GeForce GTX 780 Ti video cards. In this case, the simulated system consisted of $5 \cdot 10^7$ electrons with energy 15 keV, incident normally on a thick layer of germanium. The electron and photon cutoff energy was 1 keV. The operating system was 64-bit Windows 8.1.

Table 1. Comparison of simulation times

Program	Number of CPU threads	Mode	Number of GPUs used	Time (min)
MCNP6.1	16	CH	0	15.5
MCNP6.1	16	SE	0	54
MCNelectron v.1.1.0	16	SE	0	23.8
MCNelectron v.1.1.0	11	SE	1	15.7
MCNelectron v.1.1.0	11	SE	2	12.5

The performance gains caused by using CUDA are strongly dependent on the amount of branching in the code that is executed by the GPUs. The GPU architecture can be defined as "Single Instruction, Multiple Data streams" (SIMD), which means that in order to achieve maximum performance a majority of the CUDA threads should always perform the same instruction simultaneously, albeit on different data. This is in contrast with the "Single Instruction, Single Data stream" (SISD) architecture implemented on each core of a multi-core CPU. Because of the SISD architecture, the different cores of a CPU are completely independent and branching does not affect the performance gains caused by using multiple CPU cores simultaneously. However, the code that is executed by CUDA devices must be structured quite differently. CUDA groups all GPU threads into so-called "warps", each consisting of 32 threads. To quote "CUDA C Programming Guide" [2]: "If threads of a warp diverge via a data-dependent conditional branch, the warp serially executes

each branch path taken, disabling threads that are not on that path, and when all paths complete, the threads converge back to the same execution path." Thus, all threads of each warp should perform the same sequence of instructions in order to minimize branching (also called "warp divergence"). In the case of MCNelectron, warp divergence is reduced by breaking each particle history into "states" (e.g., one state is generation of a source particle, another one is calculation of cross section, yet another one is calculation of the distance to collision, etc.), assigning each warp a particular state, and ensuring that at every moment of time the number of particles with identical states is sufficient to completely "populate" each warp (i.e., one particle of a given state per one GPU thread of the corresponding warp). Each thread of a warp then invokes the corresponding "state handler". Since the state is the same for all threads in a warp, their state handlers perform roughly the same sequence of instructions, hence branching is reduced. The new state after executing a state handler may be different for different threads of a given warp. This necessitates re-assigning states to warps and re-distributing the particles among warps in the next iteration. This step is done serially, hence it limits the maximum performance that can be achieved. In MCNelectron v1.1.2, the mentioned re-distribution step takes about 15 % of entire processing time.

2. Cross section data files

The MCNelectron distribution package includes ENDF cross sections only for five noble gases (helium, neon, argon, krypton and xenon), as well as for oxygen, gold and uranium. They are in the subfolder "Data\ENDF". The complete set of ENDF/B electro-atomic, photo-atomic and atomic relaxation data can be downloaded from http://www.nndc.bnl.gov/endf/b7.1/download.html, as well as from several other locations. Below are the links to the three zip archives with all ENDF data that may be needed:

http://www.nndc.bnl.gov/endf/b7.1/zips/ENDF-B-VII.1-electrons.zip ,
http://www.nndc.bnl.gov/endf/b7.1/zips/ENDF-B-VII.1-photoat.zip ,
http://www.nndc.bnl.gov/endf/b7.1/zips/ENDF-B-VII.1-atomic_relax.zip .

Each of those files contains one of the three needed sublibraries of the ENDF/B-VI.8 library:

- Electro-atomic
- Photo-atomic
- Atomic relaxation

There must be three files with ENDF data for each of the chemical elements that compose the target material:

- a file with electron interaction cross sections,
- a file with photon interaction cross sections,
- a file with atomic relaxation data.

Those files must be in three subfolders of the folder specified in the input file after the keyword DIR (for a complete list of keywords, see Sections 4.1 – 4.6). The names of those subfolders are fixed: they must be "electrons", "photoat" and "atomic_relax", respectively. Format of the names of the mentioned files is also fixed. That format becomes clear after looking at those three names of files containing the ENDF data for argon:

e-018_Ar_000.endf
photoat-018_Ar_000.endf
atom-018_Ar_000.endf

(the number after the hyphen is the atomic number of the chemical element). For example, if the ENDF data are stored in the folder C:\ENDF\, then the full path names corresponding to the above-mentioned three files with the data for argon would be the following:

C:\ENDF\electrons\e-018_Ar_000.endf
C:\ENDF\photoat\photoat-018_Ar_000.endf
C:\ENDF\atomic_relax\atom-018_Ar_000.endf

Note: If the ENDF/B data were downloaded using the three links to electro-atomic, photo-atomic and atomic relaxation data given above, then it is sufficient to place those files into a chosen folder and extract them (the names of extracted files and subfolders will be as described above). If the ENDF/B data were downloaded from other locations, then the file names may be different.

The files with alternative low-energy electron interaction cross sections must have the same format as the files posted on the LXCat website (www.lxcat.net). Those files may have any names and may be stored in any location, because MCNelectron requires specification of full paths to those files for each of the chemical elements separately. The alternative low-energy electroionization and atomic excitation cross sections must be in different files. Thus, if alternative cross sections are used, there must be two additional files for each of the chemical elements: a file with alternative electroionization cross sections and a file with alternative excitation cross sections. There may be several electroionization or excitation processes defined in each of those files (the file format must be the same as format of files posted on www.lxcat.net). Instructions for retrieving cross sections from www.lxcat.net :

1) Select menu "DATA CENTER", then select the menu item "browse and download".
2) In the menu on the right, select "SCATTERING CROSS SECTION" and click "NEXT".
3) In the menu on the right, select or unselect the necessary databases and click "NEXT".
4) Select the necessary chemical element and click "NEXT".
5) Select either "Excitation" or "Ionization" and click "NEXT".
6) Ensure that all excitation or ionization processes are selected and click "NEXT".
7) Left-click the button "DOWNLOAD DATA" and save the page, or right-click that button and use the context menu item "Save Link As…". The full path of the created file will have to be specified in the file with alternative cross section information (its format is described in Section 4.2).

Note: Alternatively, data can be retrieved from www.lxcat.net by selecting the menu item "download for offline use" in Step 1. However, then the downloaded file will contain data for all interaction processes. Since MCNelectron requires the excitation and ionization cross sections to be in separate files, the file retrieved by the latter method will have to be edited with a text editor.

3. Compiling MCNelectron

MCNelectron is a Windows application (the oldest supported version of Windows is Windows XP SP2; all newer versions of Windows are supported). There is a separate distribution package (a WinRAR self-extracting archive) for each of the two mentioned versions of MCNelectron, i.e., the CPU-only version and the CUDA version. Each of those packages contains the source files, the final executable file ("MCNelectron.exe" or "MCNelectron_CUDA.exe", respectively) and a Microsoft Visual Studio 2010 project for compiling MCNelectron. The mentioned two file names correspond to the executables compiled using the "Release" configuration of Microsoft Visual Studio 2010. The names of executables compiled using the "Debug" configuration are "MCNelectron_debug.exe" and "MCNelectron_CUDA_debug.exe", respectively (the latter two files are not included in the distribution packages). The following four source files are needed for compiling the CPU-only version of MCNelectron:

MCNelectron.cpp,
RandGen.cpp,
RandGen.h,
get_executable_name.cpp.

The following seven source files are needed for compiling the CUDA version of MCNelectron:

MCNelectron_CUDA.cu,
MCNelectron_CUDA2.cu,
MCNelectron_CUDA.h,
RandGen.cu,
RandGen_CUDA.cu,
RandGen_CUDA.h,
get_executable_name.cpp.

In order to be able to compile the CUDA version of MCNelectron, the CUDA Toolkit 6.5 must be installed (it can be downloaded from https://developer.nvidia.com/cuda-toolkit-65). Note that compiling MCNelectron_CUDA does not require presence of a CUDA device in the computer. If no CUDA devices are available, the compiled code will still be able to perform CPU-only computations. At least 8 GB of available physical memory are recommended in order to be able to compile the release version of MCNelectron_CUDA on 64-bit Windows 7 or 8 (the requirements for computers running 32-bit Windows are not known, because at the time of this writing the only attempt to compile MCNelectron_CUDA on 32-bit Windows was made on a computer with 4 GB of RAM, of which approximately 2.5 GB were available, and then only the debug version of MCNelectron_CUDA was compiled successfully, whereas an attempt to compile the release version failed with a message about a memory allocation error).

The Microsoft Visual Studio 2010 project for compiling MCNelectron_CUDA produces an executable file that will run on CUDA devices with compute capabilities 2.0, 3.0, 3.5, 3.7, and 5.0. However, because of this wide range of target architectures, the compilation time is more than twice longer and the executable file is several times larger than in the case of a single compute capability. If the code is intended to be used on a particular type of CUDA devices, then the compilation time may be shortened and the size of the executable file may be decreased by specifying only the compute capability of that device in CUDA C compiler options. This is done using the Visual Studio menu command "Project/MCNelectron_CUDA Properties…". When this command is selected, a dialog window with the project property pages is opened. Then, using the left-hand pane of that dialog window, the page "Configuration Properties / CUDA C/C++ / Device" must be activated. The target architectures are entered in the cell "Code Generation". The above-mentioned set of target architectures is specified by the following line:

```
compute_20,sm_20;compute_30,sm_30;compute_35,sm_35;compute_37,sm_37;compute_50,sm_50;
```

In order to produce a code for CUDA devices with a specific compute capability, e.g., 3.5, that line must be replaced by

```
compute_35,sm_35;
```

An example of a CUDA device with compute capability 3.5 is a Nvidia GeForce GTX 780 video card. An example of a CUDA device with compute capability 2.0 is a Nvidia GeForce GTX 580 video card.

4. Running Monte Carlo simulations with MCNelectron

MCNelectron is a Win32 console application (the Windows console is opened by running the Windows command-line interpreter "cmd.exe"). In the simplest case, the command line is

MCNelectron.exe in <input_file_name> out <output_file_name>

or:

MCNelectron.exe in <input_file_name> out <output_file_name> alt <alternative_cross_sections_info_file>

(if the CUDA version is used, "MCNelectron" must be replaced by "MCNelectron_CUDA"). In the case of the first command-line format, only ENDF cross section data files and the data files contained in the MCNelectron distribution package (subfolder "Data") will be used. In the case of the second command-line format, a part of low-energy electron interaction cross sections will be read from files with alternative cross sections supplied by the user. The file with alternative cross

section info contains the names of files where those cross sections are stored, the optional interpolation information and optional additional energies (eV) for linearly interpolated values of alternative electroionization cross sections that should be inserted in the energy range where the alternative electroionization cross sections are defined. If there are no additional energies, then the final set of alternative electroionization cross sections will be defined for the same energies as ENDF electroionization cross sections.

By default, MCNelectron runs in single-thread mode on the CPU only (i.e., without using CUDA). In order to use multiple concurrent threads during simulation, their number must be specified on the command line by inserting the keyword "tasks", followed by the number of threads, for example:

MCNelectron.exe tasks 4 in input.txt out output.txt

In the latter example, four independent threads would be used on the CPU during the simulation. On multi-processor (or multi-core) systems, multithreading can significantly reduce simulation time. The number of threads can be assigned any value that does not exceed the number of streams of random numbers (see description of the keyword NSTREAMS below). However, there is no point in using a number of threads that is greater than the number of logical processors in the system (a further increase of the number of threads will not cause a decrease of the simulation time). The final results of the simulation do not depend on the number of threads that was used during the simulation. In the case of the CUDA version of MCNelectron, the number of GPU threads used by each available CUDA device can also be specified on the command line using the keyword "threads_CUDA", e.g., "threads_CUDA 960" (for more information about CUDA-specific keywords, see Section 4.6).

Another optional keyword that may be inserted in the command line is "output_cs 1" or "output_cs 0". If "output_cs 1" is used, files with values of electron interaction cross sections are created before starting the simulation. In such a case, MCNelectron creates a folder "Cross sections" in the current folder, with subfolders for each of the chemical elements. Each of those subfolders contains the following text files:

- a file with values of elastic, bremsstrahlung, excitation, total ionization, total inelastic and total interaction cross sections corresponding to the energy interval that is used during the simulation,
- files with values of ionization cross section for each electronic subshell and for all available energies of the incident electron (including the energies that are not needed for the simulation),
- a file with values of the energy loss during atomic excitation for all available energies of the incident electron.

Each of those files contains two or more columns of numbers. The first column contains electron energy values (eV). The other columns contain cross section values (b) or energy loss values (eV).

It is possible to override the specification of the source energy distribution defined in the input file by using the command-line keyword "E", followed by an energy value, (e.g., "E 10"). In such a case, a source of monoenergetic particles will be simulated. The number of source particles specified in the input file may be overridden on the command line using the keyword "N".

During the simulation, the CPU-only version of MCNelectron displays the time elapsed since the start of simulation (in seconds), the current number of finished histories, as well as the sequence number of the last source particle and the current number of banked particles for each of the first three or five active threads (those numbers are updated every second). In addition, if "control tallies" are used (see Section 4.4), the corresponding largest error is shown. MCNelectron_CUDA displays some real-time statistical information about the CUDA devices used for the simulation (for more information about the CUDA device statistics, see Section 4.7). At the end of the simulation, the contents of the output file are reproduced in the console window. The output format is similar to format of a section of a MCNP output file. All statistics under "electrons" include the contribution from positrons, too. To avoid text wrapping (which would reduce readability of the output data on the screen), the screen buffer width must be at least 170 (this can be achieved, e.g., by entering "mode 170" at the command prompt in the console window before starting the simulation). Simulation can be interrupted by pressing the keys "Ctrl+C".

4.1. Input file format

Input file example:

```
MAT   54 0.1  18 0.2  2 0.7
CONC -1
THICK 1e-5
E_UNIT MeV
CUT_E 1e-5
CUT_P 1e-6
PART E
TRACKPOS 0
DIR C:\ENDF\
N 1000
1
```

Each line of the input file, excluding the lines with the definition of the source energy spectrum (the final line in the above example), starts with a keyword. Keywords are case-insensitive. Below are descriptions of the keywords that are common to both the CPU-only and CUDA versions of MCNelectron:

MAT is used to define composition of the target material. For each chemical element, there must be two numbers: atomic number and atomic fraction. If the fractions are negative, they will be interpreted as mass fractions. No more than 20 chemical elements are allowed.

CONC is used to specify the total atomic concentration (cm^{-3}). If the number that follows CONC is negative, it is interpreted as total density (g/cm^3).

THICK is used to specify the thickness of the target layer (cm). If the number that follows THICK is negative, it is interpreted as mass thickness (g/cm^2).

SURFACE X Y Z A B C is used to specify one surface of the layer. X, Y and Z are coordinates of any one point of that surface (in cm). A, B and C are the components of the normal vector (that vector is not required to be of length 1). The other surface of the layer is in the direction of the normal vector relative to the surface defined by this keyword, at a distance defined using the keyword THICK (see above).

BEAM X Y Z A B C is used to specify the starting point and direction of the incident beam. X, Y and Z are the coordinates of the starting point (in cm). A, B and C are the components of the vector that has the same direction as the incident beam (that vector is not required to be of length 1). There are no restrictions on the values of X, Y and Z (for example, the starting point of the beam may be inside the target layer).

E_UNIT specifies the unit of energy that is used in the input and output files. It must be one of three units: “MeV”, “keV”, or “eV”.

CUT_E is used to specify the low-energy cutoff value for electrons.

CUT_P is used to specify the low-energy cutoff value for photons.

PART should be followed by the letter ‘E’ if source particles are electrons, ‘P’ for photons, or “E+” for positrons.

TASKS is used to specify the number of concurrent simulation threads on the CPU (for example, “TASKS 4”). I.e., this keyword has the same function as the mentioned command-line keyword “tasks”. If the keyword “TASKS” is specified both on the command line and in the input file, then priority is given to the number of threads specified on the command line (in such a case, a corresponding message is displayed in the console window). ***Note***: If CUDA devices are used, then one CPU thread is dedicated to communication with them and is not used for the actual simulation of particle interactions. This thread is not included in the number of threads specified after the keyword TASKS.

The following 19 keywords are used as "switches", which "turn on" or "turn off" certain features of the simulation. A non-zero value of the integer number that follows each of those keywords means that the feature is "turned on", and the zero value means that it is "turned off". Those features are:

TRACKPOS – tracking of particle coordinates (i.e., position and direction of motion),

COH – coherent scattering of photons (if TRACKPOS is 0, then the option "COH 1" is applied to source photons only),

ELASTIC – elastic scattering of electrons and positrons,

TRACK_E – tracking of secondary electrons and positrons,

TRACK_P – tracking of secondary photons,

TRACK_P_E – tracking of photon-produced electrons and positrons,

TRACK_K – tracking of knock-on electrons,

TRACK_B – tracking of bremsstrahlung photons,

TRACK_X – tracking of X-ray photons (both fluorescence and electron X-rays),

ION_DWBA – replacement of the ENDF/B inner-shell electron impact ionization cross sections by those calculated using the distorted-wave Born approximation (DWBA). The latter cross sections are the same ones that are used by the PENELOPE code system. They were calculated using the code by D. Bote, F. Salvat, A. Jablonski, and C. J. Powell, which was published in 2009 [3]. Those cross sections are stored in the file "Data\IonDWBA.dat" in binary format. ***Note***: The option "ION_DWBA 1" is not allowed when a file with alternative cross sections information is used (command-line keyword "alt").

ELASTIC_DPW – replacement of the ENDF/B electron elastic scattering cross sections by those obtained from relativistic (Dirac) partial-wave calculations. Those cross sections were calculated using the code ELSEPA by F. Salvat, A. Jablonski and C. J. Powell [4] (the same cross sections are used by the PENELOPE code system). They are stored in binary files "Data\ElasticDPW_totalCS.dat" and "Data\ElasticDPW_angularDistr.dat". The latter file contains angular distributions calculated for electron energies from 10 eV to 1 GeV on a logarithmic scale (the increment of the base-10 logarithm of energy is equal to 0.1).

BREMS_ANGULAR_DISTR – non-isotropic angular distribution of bremsstrahlung,

BREMS_BINARY_DATA – using binary bremsstrahlung angular distribution data (otherwise, the bremsstrahlung data are loaded from text files specified by the user),

BREMS_POSITRON_CORRECTION – partial suppression of positron bremsstrahlung in comparison with electron bremsstrahlung,

PE_ANGULAR_DISTR – non-isotropic angular distribution of photoelectrons,

PP_ENERGY_DISTR – non-uniform distribution of positron and electron energies during pair production,

PP_ANGULAR_DISTR – angular distribution of electrons and positrons created in pair production events,

INCOH_DOPPLER – Doppler broadening of energy distribution of incoherently scattered photons,

INCOH_SUBTRACT_BINDING_E – apply the binding energy correction to the energy of the Compton recoil electron (i.e., subtract the binding energy of the subshell from which the electron was ejected).

When TRACKPOS is 0 and the source emits photons, the geometrical parameters (defined using the keywords THICK, SURFACE and BEAM) are only used to determine if a source photon collided with an atom in the layer. All subsequent interactions in the same history are treated as though the layer was infinite in all directions. If the source emits electrons or positrons, then in the

case of TRACKPOS equal to 0 the geometrical parameters are not used at all. The option "TRACKPOS 0" should be used to calculate the number of secondary electrons when energy of the source particle is completely absorbed in the target material. For keywords COH, ELASTIC, TRACK_P_E, TRACK_K, TRACK_B and TRACK_X, a negative value has a special meaning: it means that the corresponding switch should be set to a default value (the latter may depend on values of other switches). The last eight mentioned "switches" (from "BREMS_ANGULAR_DISTR" to "INCOH_SUBTRACT_BINDING_E") control simulation of several physical effects, for which the ENDF/B-VI.8 library provides neither tabular probabilities nor an analytic prescription. For some of those effects, there are additional keywords controlling some details of the simulation. Those keywords and the corresponding simulation methods are described in Section 5 of this document.

SEED is used to specify the seed of the "type 1" random number generator (this is the mentioned Fibonacci series random number generator; the "type 2" random number generator is the XORWOW generator from the cuRAND library, which can be used only by CUDA devices). The seed may be any integer number from 0 to 30081 inclusive. The seed may be also specified on the command line (similarly to the above-mentioned keyword TASKS).

DIR is used to specify the folder with ENDF data. If that folder name does not contain the full path (i.e., if it does not start with the backslash '\' and does not contain the colon ':'), then it is assumed to be a subfolder of the folder where the MCNelectron executable is.

NSTREAMS is used to specify the number of independent streams of random numbers. For optimum performance, the number of streams should be a multiple of the number of threads. The maximum allowed number of streams is 65536. This parameter may be also specified on the command line. ***Note***: The maximum number of unique random number streams that can be generated by the Fibonacci series generator using a given seed (specified after the mentioned keyword "SEED") is 31329. Consequently, the streams with sequence numbers from 31330 to $2 \cdot 31329 = 62658$ will be generated using the next seed (i.e., the seed that exceeds the specified seed by 1), and the streams with sequence numbers from 62659 to 65536 will be generated using the seed that exceeds the specified seed by 2 (if the seed value obtained by adding 1 or 2 is greater than 30081, then it is reduced by 30082). This should be kept in mind when re-doing a simulation with a different seed: if the number of random number streams is greater than 31329 and the seeds differ only by 1, then some of the streams will be the same in both simulations. As a result, some of the simulated histories will be also exactly the same, which is usually undesirable. In order to ensure that different simulations do not re-use the same random number streams when the total number of streams exceeds 31329, the values of the seed in those simulations should differ at least by 2 or 3.

N is used to specify the number of incident particles. This parameter may be also specified on the command line.

COH_THR is used to specify the coherent scattering angular sampling threshold value. This value should not be greater than 0.001 or less than 0. A larger value may shorten the simulation when the angular distribution of photon coherent scattering is extremely peaked in the forward direction (i.e., when photon energy is very large). A smaller value leads to more accurate sampling of angular deflections in such a situation. However, a very small (e.g., zero) value of this parameter can make the rejection-based sampling algorithm very inefficient. This means that a very large number of tries may be needed before accepting a sampled value of angular deflection after a coherent scattering event.

CUT_WT and the keywords starting with "FORCE_" or "FORCING_" are related to interaction forcing (one of the variance reduction techniques) and are described in Section 4.5.

DIFFUSION_TOTAL_TO_INELASTIC_RATIO_THR controls an option to apply a diffusion model of elastic scattering of electrons at extremely low electron energies, when the elastic

scattering cross section is much larger than cross sections of all other types of electron interactions. When the diffusion model is applied, the program first generates the distance to the next inelastic interaction event (i.e., the path traveled to that event, not the displacement), and samples the electron coordinates corresponding to that event from a Gaussian distribution with a root-mean-square displacement proportional to the square root of the generated path. This reduces the "electron trapping" effect (which could otherwise slow the simulation significantly) at the expense of fidelity of the simulated electron trajectory when the electron energy is only a few electronvolts above the ionization threshold of the material, or when the energy is below the ionization threshold. The transition to the diffusion model occurs at a user-specified value of the total-to-inelastic cross section ratio. The keyword "DIFFUSION_TOTAL_TO_INELASTIC_RATIO_THR" is used to specify the mentioned threshold value of that ratio. The allowed values of that threshold are from 10 to 10^{18}. The diffusion model of elastic scattering is never applied if the distance from the electron to the nearest surface of the layer is less than the mean free path times the square root of this parameter. In order to "turn off" application of the diffusion model, this parameter must be set to any sufficiently large value (e.g., to the maximum possible value 1e18).

DIFFUSION_DIST_TO_RMSPATH_RATIO_THR controls application of the diffusion model of elastic scattering when the electron is close to the surface of the layer. The above-mentioned Gaussian distribution is only exact when the electron is at a depth d large enough to make the probability that the electron escapes from the layer before the next inelastic collision occurs practically zero. Consequently, the diffusion model of elastic scattering may introduce significant errors when estimating the moment and position of electron's escape from the layer if the electron is close to the surface. In order to decrease those errors, a single diffusion step mentioned above is replaced by a sufficiently large number of shorter diffusion steps, which are less than the root-mean-square displacement and also less than d (but still greater than the mean free path) if d is less than a user-specified number of r.m.s. displacements (the latter r.m.s. displacement is estimated as described above). This number is specified by the latter keyword. The allowed values of that number are from 2 to 10.

The following three keywords control the error introduced into the angular cumulative distribution function (CDF) of elastic scattering when some of the points in the CDF tables are removed (this removal of points is needed to speed up the process of searching for a probability value in the CDF table). The random variable that is sampled using those CDF tables is the cosine of the scattering angle. That cosine will be denoted μ.

ELASTIC_CDF_CUTOFF specifies the minimum value of the CDF, below which all points are removed (except for the initial point, which corresponds to CDF = 0 and $\mu = -1$). If at least one point is removed, then the initial value of the probability density function (PDF) is replaced by the average PDF over the interval of μ from -1 up to the value of μ corresponding to the first CDF value that is above the mentioned cutoff.

ELASTIC_CDF_STEP is the maximum allowed difference of two adjacent CDF values in the CDF table. A point can be removed only if the corresponding CDF value exceeds the previous accepted CDF value by less than this "step". However, this condition is not sufficient for removal of a point (see description of the next keyword).

ELASTIC_PDF_ERROR controls another condition that must be tested in order to determine if a point can be removed: the deviations of PDF values corresponding to the removed points from the linearly interpolated PDF (i.e., from the straight line joining the two accepted PDF values over the interval where the removed points are) must not exceed this parameter multiplied by the average PDF calculated over the same interval. I.e., this parameter may be interpreted as the maximum allowed relative error of PDF introduced by removal of points. If ELASTIC_PDF_ERROR is zero, no points will be removed, i.e., the original CDF and PDF tables will be used during the simulation. ***Note*:** This removal of points is useful only in the

case “ELASTIC_DPW 1” (see above for a description of the keyword “ELASTIC_DPW”); otherwise, it would not cause a significant improvement of performance, because the tables of elastic scattering angular PDF in the ENDF/B library are already small enough. Consequently, the following convention is used: if this parameter is entered with the minus sign, then it will be applied only in the case “ELASTIC_DPW 1”, otherwise it will be assumed to be zero. In order to force the reduction of points in the ENDF/B elastic scattering angular CDF tables, too, this parameter must be entered without the minus sign.

The 15 keywords that will be described next are related to “mapping” of various tables of sorted non-equidistant values used for cross section lookup (e.g., the table of electron energies used for selecting the angular CDF of elastic scattering corresponding to the current energy, or the table of CDF used for selecting the elastic scattering angle corresponding to the current value of probability) to “index arrays”. Each element of an index array corresponds to a particular “index value”, and all index values are equidistant (further on, each interval between two adjacent index values will be called a “bin”). Each element of the index array is equal to the sequence number of the largest element of the corresponding original table that does not exceed the corresponding index value. If the bins can be made narrow enough to accommodate no more than one value of the original table (a “breakpoint”), then the index array can significantly shorten the look-up procedure, because the sequence number of the index value corresponding to the current value of the control variable (for example, energy) is determined simply by dividing the latter value by the bin width. After that, only one comparison operation is needed in order to locate the two adjacent elements of the corresponding original table that “bracket” the current value of the control variable. In MCNelectron, each bin is allowed to contain up to 2 breakpoints, hence the maximum number of comparison operations that may be needed is 2. This mapping may be not only linear (as in the above example), but also logarithmic (which is optimal when the value of the control variable increases exponentially as a function of its sequence number), or “reverse logarithmic” (which is optimal when the deviation of the control variable from its maximum value decreases exponentially as a function of its sequence number).

There are six types of tables that can be mapped to index arrays:

1) the table of incident electron energy values used for estimating the cross sections of various types of electron interactions,
2) the table of electron energies, where each energy corresponds to a table of angular CDF of elastic scattering,
3) tables of angular CDF of elastic scattering mentioned above,
4) the table of incident electron energy values used for estimating the energy transfer during atomic excitation,
5) the table of incident electron energies, where each energy corresponds to a table of energy CDF of a knock-on electron (emitted during impact ionization),
6) tables of energy CDF of knock-on electrons mentioned above.

The latter 5 types of tables are defined for each chemical element separately. The first mentioned table is used for sampling both the chemical element and the type of the interaction. Regardless of the table type, at least two parameters must be defined: “index type” and “index exponent”. The corresponding keywords end with “_INDEXTYPE” and “_INDEXEXPONENT”, respectively. The list of 12 keywords used for specifying the index type and the index exponent for each of the mentioned six types of tables is provided below:

1) SIGMA_E_INDEXTYPE, SIGMA_E_INDEXEXPONENT
2) ELASTIC_E_INDEXTYPE, ELASTIC_E_INDEXEXPONENT
3) ELASTIC_CDF_INDEXTYPE, ELASTIC_CDF_INDEXEXPONENT
4) EXC_E_INDEXTYPE, EXC_E_INDEXEXPONENT

5) ION_E_INDEXTYPE, ION_E_INDEXEXPONENT
6) ION_SEC_E_CDF_INDEXTYPE, ION_SEC_E_CDF_INDEXEXPONENT

The index type defines the type of mapping. The types of mapping, sorted by increasing complexity, are the following:

index type = 1: linear, with at most 1 breakpoint in a bin,

index type = 2: linear, with at most 2 breakpoints in a bin,

index type = -1: logarithmic, with at most 1 breakpoint in a bin,

index type = -2: logarithmic, with at most 2 breakpoints in a bin,

index type = -3: reverse logarithmic, with at most 1 breakpoint in a bin,

index type = -4: reverse logarithmic, with at most 2 breakpoints in a bin.

If the index type is 0, then no mapping will be applied to the given type of tables, i.e., values will be found in those tables using the method of binary search.

The index exponent defines the maximum number of index values, i.e., the maximum size of the index array. The latter number is calculated as $2^{\text{index exponent}} + 1$. Having specified the index type and the index exponent, the program will test all index types from the simplest one up to the specified one (in the same order as above) in an attempt to find the simplest possible type of mapping that can be obtained without exceeding the specified maximum size of the index array (if no more than 1 breakpoint is required, then the types of mapping with 2 breakpoints will be skipped). If such type of mapping is found, the program will successively halve the size of the index array (i.e., decrease the index exponent by 1) in an attempt to find the smallest index exponent that can be used without exceeding the specified maximum number of breakpoints in a bin. If all attempts fail, index type will be reset to 0, which means that no mapping will be applied to the given type of tables, i.e., values will be found in those tables using the method of binary search.

In addition to the index type and the index exponent, the two types of tables with CDF values (i.e., tables of angular CDF of elastic scattering and tables of energy CDF of knock-on electrons) need one more parameter specified. That parameter indicates if the first interval in the table (i.e., the interval between the first two points) should be excluded from mapping. If so, then before starting the lookup process the program will check if the current value of the control variable belongs to that interval, and the index array will be used only if it is determined that the value is outside that interval. The corresponding two keywords are

ELASTIC_CDF_EXCLUDEFIRST and ION_SEC_E_CDF_EXCLUDEFIRST

Those two keywords must be followed by "1" or "0", depending on whether the first interval should be excluded or not. ***Note*:** These keywords have an effect only in the case of linear mapping (in the case of logarithmic or reverse logarithmic mapping of CDF tables, the first interval is always excluded).

The last parameter related to index arrays is a "switch" that controls output of information about used index arrays (i.e., types of mapping and maximum sizes of index arrays for each type of tables and each type of mapping) in the console window before starting the simulation. The corresponding keyword is

OUTPUT_INDEX_ARRAY_INFO

(it must be followed by "1" or "0", as for all other switches).

The following keywords are related to pulse-height and plane-crossing tallies:

PULSE_HEIGHT is used to specify calculation of the pulse-height tally, i.e., the tally of absorbed energy. It must be followed by three space-delimited numbers: the lower bound of the energy range, the upper bound of the energy range, and the number of bins. Then MCNelectron will divide the specified energy interval into the specified number of equidistant bins and count the number of histories when the energy absorbed in the target material belonged to each of those

bins. Each bin count is incremented when the absorbed energy is less than or equal to the high-energy limit of the bin and greater than the low-energy limit of the bin. The histories when the absorbed energy was less than or equal to the low-energy bound of the first bin, or greater than the high-energy limit of the last bin, are not counted. The tally data are placed into a text file "PulseHeight.txt", which is in a subfolder created in the same folder as the main output file. The subfolder name includes the output file name (the subfolder name format is "Files_<output_file_name>"). The file "PulseHeight.txt" contains three columns of numbers: the high-energy limits of all bins, the corresponding number of counts and the relative standard deviation of each number (the relative standard deviation is equal to the inverse square root of the number of counts). ***Note***: It is also possible to specify the "target" relative standard deviation that must be reached in order to stop the simulation (see Section 4.4 for more details).

ETALLY or ETALLY<n>, where "<n>" is an integer number from 1 to 20, specifies a plane or a set of parallel equidistant planes, as well as one or more tallies of the number of electrons or positrons crossing those planes (see Section 4.3 for more information about the tallies).

PTALLY or PTALLY<n>, where "<n>" is an integer number from 1 to 20, specifies a plane or a set of parallel equidistant planes, as well as one or more tallies of the number of photons crossing those planes (see Section 4.3 for more information about the tallies).

E, THETA, MU, R, PHI, X, or Y, or the same keyword followed by an integer number from 1 to 20 (without a space), specifies a range of values of one of seven quantities (energy, angle of incidence, cosine of the incidence angle, radial coordinate, azimuthal angle, X coordinate, or Y coordinate respectively) to be used in specifications of plane-crossing tallies (see Section 4.3 for more information about plane-crossing tallies and about defining a range of values).

TALLIES_PER_SOURCE is a "switch" that specifies whether the counts in pulse-height and plane-crossing tallies should be divided by the total number of source particles.

E_CONTIGUOUS is a "switch" that specifies the order of energy and probability values in the definition of the source energy spectrum (see below).

Any non-empty line that starts not with one of the mentioned keywords is interpreted as a part of the definition of the energy spectrum of incident particles. Such lines must contain only numbers. If there is only one such line and if it contains only one number (as in the above example), then a source of monoenergetic particles will be simulated, and that number will be interpreted as the source energy value. In the above example, the source emits 1000 electrons with energy 1 MeV. If there are two or more numeric values in total, then one half of them should have the meaning of energy and the other half should have the meaning of corresponding weights. Each weight is proportional to probability that the source particle energy will belong to the energy interval with limits equal to the last two energy values (i.e., those numbers define the table of the particle energy distribution). The mentioned weights need not to be normalized to 1 (MCNelectron does that automatically). The first weight must be zero. The relative order of energies and weights is defined using the mentioned keyword "E_CONTIGUOUS". If the integer number that follows E_CONTIGUOUS is non-zero, then the first half of all numbers will be interpreted as energy values, and the second half will be interpreted as the weights, listed in the same order as energies. Otherwise, energies and corresponding weights must be in pairs: "Energy No.1 Weight No.1 Energy No.2 Weight No.2", etc. ***Note***: In order to make it easier to copy spectrum data from MCNP input files into MCNelectron input files, MCNelectron ignores the character '&', which is used as a line-continuation character in MCNP input files (in MCNelectron input files, a line continuation character is not required).

Only the definition of the source energy spectrum and the keywords MAT, CONC, THICK, N must be present in the input file. All other keywords may be absent. Then the corresponding parameters will be assigned default values. The keyword THICK may be absent when TRACKPOS is 0 and the source emits electrons or positrons, because in that case the geometrical parameters are

not used during simulation. In addition, if the source energy value and the number of source particles have been specified on the command line using the keywords "E" and "N", then definition of the source energy spectrum and the keyword "N" may be absent, too (otherwise those specifications in the input file will be ignored). The default values are the following:

```
SURFACE       0 0 0   0 0 1
BEAM          0 0 0   0 0 1
E_UNIT        MeV
CUT_E         1e-5
CUT_P         1e-6
TASKS         1
PART          E
All 19 switches related to the physical models are "on" (i.e., equal to 1)
TALLIES_PER_SOURCE      0
E_CONTIGUOUS      1
SEED          123
DIR           C:\ENDF\
NSTREAMS      64
COH_THR       0.0001
DIFFUSION_TOTAL_TO_INELASTIC_RATIO_THR      100
DIFFUSION_DIST_TO_RMSPATH_RATIO_THR  5
ELASTIC_CDF_CUTOFF      1e-12
ELASTIC_CDF_STEP        0.01
ELASTIC_PDF_ERROR       -0.1
SIGMA_E_INDEXTYPE       -2
SIGMA_E_INDEXEXPONENT       17
ELASTIC_E_INDEXTYPE     -2
ELASTIC_E_INDEXEXPONENT     17
ELASTIC_CDF_INDEXTYPE       -2
ELASTIC_CDF_INDEXEXPONENT       18
EXC_E_INDEXTYPE         -2
EXC_E_INDEXEXPONENT   17
ION_E_INDEXTYPE         -2
ION_E_INDEXEXPONENT   17
ION_SEC_E_CDF_INDEXTYPE     -4
ION_SEC_E_CDF_INDEXEXPONENT     18
ELASTIC_CDF_EXCLUDEFIRST    1
ION_SEC_E_CDF_EXCLUDEFIRST      0
OUTPUT_INDEX_ARRAY_INFO     0
```

The default values of the parameters related to interaction forcing are given in Section 4.5.

The subfolder "Test" of the distribution package contains input files for MCNelectron and MCNP6 corresponding to identical simulation conditions, along with respective output files, which demonstrate a very good agreement between the results obtained with both those programs (instructions for doing those simulations and additional information are in the file "Test_Test_Info_.pdf" and in Section 7 of this document).

4.2. Format of the file with alternative cross section information

An example of the file with alternative cross section information:

```
EXC_FN 18 c:\Plasma Data Exchange Project\e-Ar excitation - SIGLO.txt
EXC_INTERP_END 18 5e5
EXC_INTERP_RULE 18 5
EXC_INTERP_COEFS 18 12.3276 2.08644 -0.273936 0.00864765
```

```
ION_FN 18 c:\Plasma Data Exchange Project\e-Ar ionization - SIGLO.txt
ION_INTERP_END 18 1e5
ION_INTERP_RULE 18 5
ION_INTERP_COEFS 18 26.0849 -0.962273 -0.035216 0.00279351

EXC_FN 54 c:\Plasma Data Exchange Project\e-Xe excitation - SIGLO.txt
EXC_INTERP_END 54 5e5
EXC_INTERP_RULE 54 5
EXC_INTERP_COEFS 54 31.0399 -3.8101 0.32736 -0.0103518
ION_FN 54 c:\Plasma Data Exchange Project\e-Xe ionization - SIGLO.txt
ION_INTERP_END 54 1e5
ION_INTERP_RULE 54 5
ION_INTERP_COEFS 54 15.3521 2.44828 -0.370784 0.0135908

70
80
90
100
110
120
130
```

Each line of the file with alternative cross section information, excluding the lines with additional energies, starts with a keyword, which is followed by the atomic number (Z) of a chemical element. Explanation of the keywords used in that file:

EXC_FN is used to specify the file name with alternative low-energy excitation cross sections. In the above example, that file name is specified for Ar and Xe (Z = 18 and 54).

EXC_INTERP_END is used to specify the high-energy endpoint of the interpolation range (eV) for excitation cross sections. In the above example, that endpoint is 500 keV. ***Note***: The low-energy endpoint of the interpolation range coincides with the largest value of the electron energy in the file with alternative cross sections. If the specified high-energy interpolation endpoint is less than the just-mentioned energy, then there will be no interpolation.

EXC_INTERP_RULE is used to specify interpolation rule for excitation cross sections. Four interpolation rules are supported: linear-linear, linear-log, log-linear and log-log. They are denoted by integer numbers from 2 to 5 (by analogy with ENDF format):

2 – linear-linear, 3 – linear-log, 4 – log-linear, 5 – log-log.

EXC_INTERP_COEFS is used to specify four coefficients of the third-degree polynomial that is used for interpolation of excitation cross sections. Those coefficients must be listed in the order of increasing degree: the first coefficient corresponds to the term of degree zero, and the last coefficient corresponds to the term of degree 3. In the above example, the natural logarithm of the excitation cross section for argon in the interpolation range is calculated as follows:

$\ln(\sigma) = 12.3276 + 2.08644 \cdot \ln(E) - 0.273936 \cdot [\ln(E)]^2 + 0.00864765 \cdot [\ln(E)]^3$,

where the energy E is expressed in eV, and the cross section σ is expressed in barns.

Keywords ION_FN, ION_INTERP_END, ION_INTERP_RULE, ION_INTERP_COEFS are used to specify the files with alternative ionization cross sections and the corresponding interpolation information. Their usage is the same as for excitation cross sections (see above).

If no interpolation is needed, the lines starting with “EXC_ INTERP_” or “ION_INTERP_” may be omitted. Any non-empty line that starts not with one of the mentioned keywords is interpreted as a part of the list of additional energies for calculation of ionization cross sections. Each such line must contain only one energy value (eV), and all those energies must be less than the largest energy in all files with alternative ionization cross sections. If there are no additional energies, then the values of alternative ionization cross sections will be calculated for the same energies that are given

in the ENDF tables of ionization cross sections for each electronic subshell, using linear interpolation. The set of additional energies is the same for all chemical elements that compose the target material. ***Note***: The additional energies should be used when the maximum of the ionization cross section is not represented well enough in ENDF tables of ionization cross sections (because of too large intervals between energy values in the ENDF electroionizaton data).

4.3. Plane-crossing tallies

MCNelectron can calculate tallies of electrons or photons that have crossed a user-defined "tallying plane" or a set of parallel equidistant tallying planes during the simulation. The tally parameter (i.e., the variable whose value determines the sequence number of the incremented bin of the tally) can be any one of those seven quantities:

1) the particle energy,
2) the angle between the plane's normal and the particle's momentum vector,
3) the cosine of the angle between the plane's normal and the particle's momentum vector,
4) the radial coordinate of the crossing point on the tallying plane,
5) the azimuthal angle of the crossing point on the tallying plane,
6) the X' coordinate of the crossing point on the tallying plane,
7) the Y' coordinate of the crossing point on the tallying plane.

The latter four variables are defined in the tallying plane's coordinate system, whose axes are in general different from the axes of the system that is used for specifying position of the target layer and the incident beam direction. That is why there is the prime symbol in notations X' and Y'. It is also possible to calculate "two-dimensional" tallies, where each bin corresponds to a pair of intervals of any two variables (for example, the particle energy and the radial coordinate). All bins are of equal width.

The specification of a plane-crossing tally contains notations of ranges of the variables that must be used for calculating the "scoring" bin. Each of those ranges must be defined before the specification of the corresponding tally in the input file. Definition of a range must begin with any one of those seven keywords:

"E", "THETA", "MU", "R", "PHI", "X", or "Y",

or the same keyword with an appended number from 1 to 20 (for example, "E1", "MU10", etc). This keyword defines the meaning of the variable (the energy, the incidence angle, the cosine of the incidence angle, the radial coordinate, the azimuthal angle, the X' coordinate, or the Y' coordinate, respectively). After that keyword, there must be three space-delimited numbers: the lower bound of the range, the upper bound of the range, and the number of bins. For example, the definition of an energy range consisting of 10 equidistant bins from 1 MeV to 2 MeV might look like this:

```
E1   1 2 10
```

In this example, the first bin corresponds to energies from 1.0 MeV to 1.1 MeV, and the last bin corresponds to energies from 1.9 MeV to 2.0 MeV (the unit of energy is the same one as for all other energy values, i.e., it is defined by the keyword "E_UNIT"). The values of the incidence angle "THETA" and the azimuthal angle "PHI" must be given in degrees.

The specification of a plane-crossing tally for electrons or positrons must begin with the keyword "ETALLY", or the same keyword with an appended number from 1 to 20 (for example, "ETALLY2"). The specification of a plane-crossing tally for photons must begin with the keyword "PTALLY", or the same keyword with an appended number from 1 to 20 (for example, "PTALLY2"). That keyword must be followed by six space-delimited numbers: three coordinates of a point on the tallying plane (in cm) and three components of the normal vector of the tallying plane (that vector is not required to be of length 1). Those numbers must be followed by space-delimited notations of ranges that have been previously defined in the input file. An example of the simplest possible tally:

```
ETALLY1   0 0 0   0 0 1   E1
```

The above example defines an energy tally of electrons or positrons crossing the XY plane. During the simulation, each bin count will be incremented when an electron or a positron crosses that plane

and it is determined that the value of the control variable (energy in the above example) is less than or equal to the upper limit of that bin and greater than the lower limit of that bin. If the control variable is not the incidence angle or its cosine ("THETA" or "MU", respectively), the count is increased by 1 when the tallying plane is crossed in the direction of its normal vector, and decreased by 1 when the tallying plane is crossed in the opposite direction. If the control variable is the incidence angle or its cosine, then the count is always increased by 1, because the value of that variable already indicates the direction of crossing.

In order to specify a set of parallel equidistant tallying planes, two additional space-delimited numbers must be inserted before the list of ranges: the number of planes and the inter-plane distance (in cm). For example, the following line defines 100 tallies, each corresponding to one of 100 planes, which are at a distance of 0.1 cm from each other:

ETALLY1 0 0 0 0 0 1 100 0.1 E1

In the latter case, the first three numbers define a point that is on the *first* plane in the set. All other planes in the set are "stacked" in the direction of the normal vector (defined by the fourth, fifth and sixth numbers).

It is possible to group any two ranges in the specification of a plane-crossing tally by enclosing their notations with parentheses, for example:

ETALLY1 0 0 0 0 0 1 100 0.1 E1 (X1 Y1) (E1 MU2)

Each such pair of ranges corresponds to a two-dimensional tally, which is represented by a table with rows numbered by the first (or "primary") variable of the pair, and with columns numbered by the second (or "secondary") variable of the pair. There can be up to 20 one-dimensional tallies and up to 20 two-dimensional tallies defined in a single specification of a plane-crossing tally. Thus, the maximum possible number of tallies defined in a single line of the input file is 40 (multiplied by the number of planes in the "stack").

Any of the mentioned seven variables can be limited by specifying its bounds (i.e., minimum and maximum values) in the definition of a tally. This is achieved using the keyword "LIM", followed by notation of the variable ("E", "THETA", "MU", "R", "PHI", "X" or "Y"), followed by two space-delimited values of the bounds. Those four components may be inserted anywhere after the geometric parameters (i.e., after the first six numbers in the case of a single-plane tally, or after the first eight numbers in the case of a multi-plane tally). For example, in the case of the following tally, "scoring" will be possible only for electrons and positrons that cross a ring-shaped area with inner and outer radii equal to 10 cm and 12 cm, respectively, and with energies between 0.05 MeV and 0.1 MeV:

ETALLY1 0 0 0 0 0 1 100 0.1 LIM R 10 12 (MU1 PHI1) LIM E 0.05 0.1

Those limits apply to all tallies defined on the same line of the input file. The lower bound is excluded from allowed values, whereas the upper bound is included in allowed values.

Usage of the variables "R", "PHI", "X", and "Y" requires defining the coordinate system on the tallying plane (i.e., position of the origin and directions of the coordinate axes X' and Y'). The origin of that coordinate system is the same point that is specified by the first three numbers in the specification of a tally (or projection of that point for other planes in the "stack"). If the tallying plane is not normal to the X axis of the "main" coordinate system, then direction of the X' axis is the same as direction of the projection of the "main" X axis to the tallying plane. This means that if the tallying plane is not parallel to the X axis, then direction of the X' axis is the same as direction of the vector with components (A', B, C), where $A' = A - (1 / A)$ and A, B, C are the components of the tallying plane's normal vector with length 1. If the tallying plane is normal to the X axis of the main coordinate system, then the X' axis is the one that is either the same direction as the Y axis of the main coordinate system (if the normal is directed in the positive X direction), or opposite to the Y axis (if the normal is directed in the negative X direction). In any case, direction of the Y' axis on the tallying plane is such that the X' axis, the Y' axis and the normal vector of the tallying plane are oriented relative to each other according to the right-hand rule (for example, if the tallying plane is normal to the X axis, then, according to the previously-defined direction of the X' axis, the direction of the Y' axis is the same as the direction of the Z axis of the "main" coordinate system). Having

defined the coordinate system on the tallying plane, the radial coordinate "R" can be defined as the distance from the point of crossing to the origin of that coordinate system, and the azimuthal angle "PHI" can be defined as the angle between the *X'* axis and the line joining the origin with the point of crossing. The positive direction of measuring the azimuthal angle "PHI" is counterclockwise when observed from the half-space where the normal vector is pointing to.

Each tally data are written to a separate text file. Those files are in the same subfolder as pulse-height tally data files (the subfolder name format is "Files_<output_file_name>"). For single-plane tallies, the file name consists of the tally and range identifiers, separated by an underscore, e.g., "ETALLY1_E1.txt" in the case of a 1D tally, or "ETALLY1_X1_Y1.txt" in the case of a 2D tally. For multi-plane tallies, the file name also includes the sequence number of the plane in the set of parallel planes and the distance between that plane and the first plane of the set (in cm), which is written in parentheses immediately after the plane number, e.g., "ETALLY1_30(2.9)_E1.txt". In each file, the first line contains column headers. All other lines contain upper bounds of the bins of the primary variable in the first column and the corresponding particle counts and relative standard deviations in the other columns. If the input file contains the option "TALLIES_PER_SOURCE 1", then the counts will be per source particle (i.e., they will be divided by the total number of source particles). For 1D tallies, the header of the second column is "N", and the header of the third column is "RelSDev". For 2D tallies, headers of all columns with particle counts or relative standard deviations have the format "N_<value>" or "D_<value>", respectively, where "<value>" is the upper bound of the corresponding bin of the secondary variable. The column headers and all numerical values in tally data files are tab-delimited.

If no interaction forcing is used (see Section 4.5), then the (absolute) standard deviation of the particle count in each bin is equal to the square root of the number of particles that have crossed the given plane in any direction, subject to the constraints of the given bin. This is equivalent to assuming that all particles counted in a given bin are independent and have weights equal to 1. The relative standard deviation is obtained by dividing the mentioned absolute standard deviation by the absolute value of the particle count corresponding to the same bin (note that the latter particle count may have been calculated with different signs assigned to particles crossing the plane in different directions). If interaction forcing is used, then particles that are counted in each bin have in general different weights. In this case, the absolute standard error is calculated as the square root of the sum of squared weights of all particles that have been counted in the given bin, regardless of their direction of crossing.

4.4. Specifying the maximum error for plane-crossing and pulse-height tallies

For each plane-crossing or pulse-height tally, it is possible to specify the maximum relative standard deviation (or "target relative error"). If it is determined during the simulation that the relative standard deviation for each of those tallies is less than the corresponding target error, the simulation will be terminated immediately. Thus, the target errors are used as alternative criteria for determining if a simulation may be stopped. The tallies with specified target errors will be therefore called "control tallies". The simulation is stopped when the target error is reached for each of the control tallies, or when all source particles are processed (the number of source particles is specified in the input file or on the command line after the keyword "N"). The target error is specified using one of the following three keywords:

AVGERR
MINERR
MAXERR

That keyword must be followed by a value of the target error. In order to determine if the target is met, the program will periodically compare the average, minimum or maximum relative standard deviation, calculated over all bins of all tallies defined on the same line, with the corresponding target error (this comparison is done once every second). In the case of a plane-crossing tally, that keyword may be inserted anywhere after the geometric parameters (i.e., after the first six numbers

in the case of a single-plane tally, or after the first eight numbers in the case of a multi-plane tally), for example:

ETALLY1 0 0 0 0 0 1 100 0.1 E1 AvgErr 0.05

In this example, the simulation will be stopped when the standard deviation averaged over all bins in all 100 tallies of the set becomes less than 5 % (assuming that there are no other control tallies defined). In the case of a pulse-height tally, one of the mentioned three keywords must be added at the end of the tally specification, for example:

PULSE_HEIGHT 0 1 100 MaxErr 0.02

In this example, the simulation will be stopped when the maximum standard deviation over all 100 bins of this pulse-height tally becomes less than 2 %.

During the simulation, the program displays the current value of the relative error for the control tally and the corresponding target error. If there are two or more control tallies, then the displayed error is the one that is furthest from the target error (in relative terms), i.e., that has the largest ratio to the target error. For example, in the case of the two tallies defined above, the message that is appended to the line of text with second-by-second statistics shown in the console window might look like this:

```
AvgErr (ET1) = 9.4% / 5%
```

or:

```
MaxErr (PH1) = 2.82% / 2%
```

In those messages, the characters in parentheses indicate the type of the tally and its sequence number among all tallies of the same type (in the above examples, those are the plane-crossing tally No. 1 for electrons, i.e., "ETALLY1", and the pulse-height tally No. 1).

4.5. Interaction forcing

Interaction forcing is a variance reduction technique that is based on an artificial increase of frequencies of certain types of particle interactions. The bias that could be caused by this method is avoided by assigning weights to particles such that the sum of the weights of all secondary particles produced in a certain type of interactions is equal to the number of secondary particles that would be produced in the same interactions during an analog simulation (the "analog" simulation is the one that does not apply variance reduction techniques).

In principle, the following approach could be applied in order to take into account all possible interactions (even the ones with extremely small probability) without the need to simulate a large number of histories (one "history" is the sequence of all interaction events started by a single source particle):

1. Whenever a collision occurs, split the particle into as many particles ("replicas") as there are interaction types and assign the weight to each "replica" equal to the weight of the original particle (W) multiplied by the relative frequency of the corresponding interaction type. Since the sum of all relative frequencies is equal to 1, the sum of all those weights is equal to W.
2. Make all those "replicas" identical to the original particle in all other respects (i.e., the same energy, the same direction of motion and the same position).
3. Simulate each type of interaction (one type of interaction for each "replica") in the usual way, i.e., sample energies and directions of the secondary particles from the relevant probability distributions.
4. Assign weights to the secondary particles equal to the weight of the corresponding "replica" of the original particle.

After that, each of the particles produced by each type of interaction (possibly including the primary particle with a changed energy or direction) is tracked by repeating the same steps. In order to obtain an unbiased value of the tallied quantity (e.g., the total energy loss, or the total number of particles produced in a certain type of interactions, or the total count in a particular bin of a particular tally), the contribution of each particle to that sum must be multiplied by the particle weight. However, this method has two obvious drawbacks:

a) Some of the interaction types may be already frequent enough, or those interactions may be not important for the simulated phenomenon (for example, elastic scattering of electrons is not important when simulating electron-induced characteristic X-ray emission). In those cases, no significant improvement in accuracy would be achieved by forcing those interactions.
b) Since each interaction is accompanied by particle "splitting", such an approach would cause an exponential growth of the total number of particles that have to be tracked, resulting in prohibitive memory requirements. In addition, the simulation time per history may become so long that it would offset any gains caused by a decrease of the number of histories that have to be simulated.

The first of the mentioned drawbacks can be eliminated by forcing only the types of interactions that are both relatively infrequent and important for the investigated phenomenon. For example, the type of interaction that is most important for simulations of electron-induced characteristic X-ray emission is inner-shell impact ionization, which is much less frequent than outer-shell impact ionization. All other types of interactions are sampled in the usual way, i.e., by selecting one of them randomly. This decreases the number of particles that have to be tracked, because all non-forced interactions are represented by just one "replica" per collision. However, the second drawback is not eliminated, because there are still at least two "replicas" per collision, and their energies are equal to energy of the original particle. The latter equality means that each of the "replicas", if tracked in an analog manner, would create, on the average, the same number of secondary particles as the original particle (although with a smaller weight). Thus, a single collision may double the total number of particles that have to be tracked. If this "doubling" is applied to all subsequent collisions, too, then the total number of tracked particles will grow in a geometric progression, slowing down the simulation and eventually causing memory overflow.

One of the ways to deal with this proliferation of tracked particles is to define a "weight cutoff", i.e., the minimum weight such that all particles with a smaller weight are removed. The growth of the number of particles is accompanied by a decrease of their weights, so that they eventually fall below the weight cutoff. However, it is difficult to determine beforehand the optimum value of the weight cutoff, i.e., such a value that would decrease the simulation time significantly without introducing a noticeable bias into the simulation results.

The default approach applied in MCNelectron is based on decreasing the average frequency of collisions where interaction forcing is applied, and a corresponding increase of the weight of the "replicas" participating in forced interactions. This is achieved by defining probability of interaction forcing ("forcing probability") P_f, and by generating an additional uniform random number before each collision. If that number is less than P_f, then the particle "splitting" is applied to the given collision as described above. Otherwise, "splitting" is not applied and one of non-forced interactions is selected randomly (in this case, particle weights are not modified). In the case of a collision with interaction forcing, the mentioned "natural" weights of all "replicas" participating in forced interactions are increased by a factor equal to $1 / P_f$ in order to compensate for the decreased frequency of interaction forcing. Thus, if the sum of the "natural" relative frequencies of all forced interactions is w_f and the weight of the original particle is W, then the sum of weights of all replicas participating in forced interactions is $W \cdot w_f / P_f$, whereas the replica corresponding to the randomly selected non-forced interaction is assigned the weight equal to $W(1 - w_f / P_f)$. The forcing probability P_f is re-calculated before each collision using the formula $P_f = w_f \cdot F$, where F is a "forcing factor" F defined by the user (F must be 10 or greater; its default value is 10). After that, P_f is compared with maximum and minimum allowed values of the forcing probability (also defined by the user), P_{fmax} and P_{fmin}. If $P_f < P_{fmin}$, then P_f is replaced by P_{fmin} (the largest possible value of P_{fmin} is 0.01, and the default is 10^{-4}). If $P_f > P_{fmax}$, then the set of forced interactions is reduced by changing the status of the interactions from "forced" to "non-forced" one by one, starting from the interaction with the highest relative frequency, until the "natural" total relative frequency of all remaining forced interactions, multiplied by the forcing factor, becomes less than the maximum allowed forcing probability (its default value is 0.01). In this case, a corresponding warning is displayed in the console window. This removal of forced interactions is temporary; all those steps

are repeated before each collision (however, the mentioned warning is displayed only once for each type of forced interactions in each CPU thread and in each used CUDA device).

As it is obvious from the above, the implementation of interaction forcing in MCNelectron ensures that the total relative frequency (i.e., total weight) of all forced interactions (w_f), adjusted to compensate for the decreased frequency of interaction forcing, i.e., the value of w_f / P_f, is much less than 1 in each collision. The largest possible value of w_f / P_f is equal to the inverse forcing factor, i.e., to $1 / F$. Since the minimum possible value of F is 10, w_f / P_f can not exceed 0.1. It is also obvious that this method increases only the interaction frequencies relative to other (non-forced) interactions, but does not change the mean free path (the average distance to collision).

The described variant of interaction forcing in some situations allows reducing simulation time needed to achieve the required accuracy in the tallies. For example, interaction forcing causes a significant decrease of the simulation time (sometimes more than by an order of magnitude) when simulating electron-induced characteristic X-ray emission. In MCNelectron v1.1.2, interaction forcing can be applied only to electron and positron interactions. There are six types of electron and positron interactions that can be forced:

1) elastic scattering,
2) bremsstrahlung,
3) atomic excitation,
4) K-shell impact ionization,
5) L-shell impact ionization,
6) M-shell impact ionization.

L-shell impact ionization is actually a set of three interactions (one for each of the three subshells of the L shell), and M-shell impact ionization is a set of five interactions (one for each of the five subshells). Consequently, if M-shell impact ionization is the only type of forced interaction, then the primary electron will be split into six replicas (five replicas corresponding to the five forced interactions and one replica for the single randomly selected non-forced interaction).

Below are descriptions of all keywords that control interaction forcing.

FORCING_FACTOR is used to specify the mentioned “forcing factor” (F).

FORCING_PROB_MIN is used to specify the minimum allowed value of the mentioned “forcing probability” (P_{fmin}).

FORCING_PROB_MAX is used to specify the maximum allowed value of the mentioned “forcing probability” (P_{fmax}).

FORCING_WT_MIN is used to specify the minimum weight of an electron or a positron that must be exceeded in order to apply interaction forcing. Particles with smaller weights can not be split into “replicas” as described above. Instead, they are tracked in an analog manner. If the particle energy (E) exceeds the minimum electron binding energy of the target material (B), then the mentioned minimum weight is additionally reduced by multiplying it by the ratio B / E. That is to say, the interaction forcing algorithm is applied to such particles, if their weight is greater than FORCING_WT_MIN $\cdot B / E$.

CUT_WT is used to specify the minimum sum of weights of all “replicas” corresponding to the forced interactions that must be exceeded in order to apply interaction forcing. If the mentioned total weight is less than this threshold, then the next collision will be simulated in an analog manner.

Caution should be exercised when modifying FORCING_WT_MIN and CUT_WT. Although an increase of those parameters may decrease simulation time, it may also introduce bias into the simulation results. For this reason, the default values of FORCING_WT_MIN and CUT_WT are zero, i.e., those thresholds are not applied by default. The remaining keywords related to interaction forcing, which are listed below, are used as “switches”, which “turn on” or “turn off” forcing of certain types of interactions (a non-zero value of the integer number that follows each of those keywords means that the corresponding interaction type will be forced, and the zero value means that it will not be forced):

FORCE_ELASTIC – forcing of elastic scattering from all types of atoms,

FORCE_BREMS – forcing of bremsstrahlung from all types of atoms,

FORCE_EXC – forcing of excitation of all types of atoms,

FORCE_ION_K – forcing of K-shell impact ionization of all types of atoms,

FORCE_ION_L – forcing of L-shell impact ionization of all types of atoms,

FORCE_ION_M – forcing of M-shell impact ionization of all types of atoms.

Note: The specified interaction will be forced only if its cross section is non-zero.

In addition, it is possible to specify interaction forcing only for a particular chemical element. This is done by appending the atomic number to any of the previous six keywords. For example, the keyword FORCE_ION_M92 should be used to turn on or off M-shell impact ionization for uranium. If both the keyword with the atomic number and the keyword without it are present, priority is given to the keyword with the atomic number. For example, if the input file contains the directives "FORCE_ION_L 1" and "FORCE_ION_L8 0", then the program will force L-shell impact ionization of all atoms excluding oxygen.

The default values of the mentioned parameters related to interaction forcing are the following:

FORCING_FACTOR	10
FORCING_PROB_MIN	0.0001
FORCING_PROB_MAX	0.01
FORCING_WT_MIN	0
CUT_WT	0

All switches are "off" (zero), i.e., interaction forcing is not applied by default.

The above-mentioned default values of parameters F, P_{fmin} and P_{fmax} have been successfully tested in simulations of electron-induced characteristic X-ray emission. However, there is no guarantee that they will work as well in other kinds of problems.

4.6. CUDA-specific keywords

There are 23 keywords that control usage of computer resources by the GPUs and the division of workload between the CPU and the GPUs. Before describing those keywords, several terms must be defined. The term "task" will be used to mean the part of the code that simulates a particle history. During the simulation, each task generates source particles and follows each source particle together with all secondary particles until all of them either escape from the layer or are lost during various interactions. Then that task generates the next source particle, etc. On a lower level (i.e., the level of hardware and the operating system), simulation is run by independent "threads", whose number is limited by the available computing resources. In the case of Nvidia GPUs, the maximum possible number of threads per one CUDA device is 1024. As explained in Section 1.3, the CUDA version of MCNelectron does not assign a fixed set of tasks to each thread. Instead, each task is done in steps by various threads, i.e., the tasks are frequently "redistributed" among the threads. For maximum performance, the number of tasks should be much larger than the number of active threads, because then there is a greater freedom for choosing the optimum set of tasks to run at each given moment of time (the mentioned "optimum" set of tasks is the one that minimizes warp divergence, as explained in Section 1.3). In contrast, each thread on the CPU follows a particle history in its entirety, i.e., each CPU task is executed by a single thread. Hence, in the case of a CPU-only simulation, the terms "task" and "thread" may be used interchangeably. The maximum possible total number of tasks (including the tasks on all used GPUs and on the CPU) is 65536.

For each history, a task uses a stream of random numbers generated from a fixed "seed". The total number of streams of random numbers is defined using the mentioned keyword "NSTREAMS" (see Section 4.1). This number includes both the streams that are used on the CPU and the streams that are used by the CUDA devices. Ideally, all streams should not be correlated with each other and should not repeat themselves. Each task is assigned a fixed set of streams and

then cycles between them for different histories (thus, the streams of random numbers are assigned to tasks, not to threads). The maximum possible total number of streams is 65536.

The part of an application that runs on a CUDA device is called a "kernel". On each CUDA device, one kernel is executed at a time, but many threads execute the kernel simultaneously. On the application level, those threads are grouped into "blocks", which are not synonymous with "warps" mentioned in Section 1.3. A block may span several warps, or it may be only a part of a single warp. In absence of warp divergence, the optimum size of a block is a multiple of the warp size, i.e., 32. If warp divergence is significant, better performance may be sometimes achieved using block size less than 32 (however, the block size should still be a multiple of 8). The results of testing indicate that the best performance on desktop computers with GeForce GTX 780 Ti and GTX 580 cards is usually achieved when the block size is 8, and in the case of a laptop computer with a GeForce GTX 960M card the best performance has been achieved using block size 32.

All CUDA-specific keywords end with "_CUDA". As all other keywords, they must be followed by the value of the corresponding parameter. 19 of the 23 CUDA-specific keywords may be specified either on the command line or in the input file. The remaining 4 keywords may be specified only on the command line. If a keyword is specified both on the command line and in the input file, then the parameter value specified on the command line will be used. Below is the list of the mentioned 19 keywords:

rand_CUDA is used to specify the type of the random number generator to be used on the CUDA devices. The number 1 indicates the Fibonacci series random number generator that is used on the CPU (in this case, the parameter SEED in the input file will also be used to initialize the instances of that generator that run on the CUDA devices). The number 2 indicates the XORWOW generator from the cuRAND library. Simulations using the latter generator are usually faster by up to 10 %;

seed_CUDA is used to specify the seed of the XORWOW generator from the cuRAND library (this parameter is used only with the option "rand_CUDA 2", otherwise it is ignored). This seed must be positive and less than 2^{64};

tasks_CUDA is used to specify the number of tasks per one CUDA device. If that number is zero, then CUDA devices will not be used and all CUDA-specific keywords will be ignored;

threads_CUDA is used to specify the number of threads per one CUDA device (it has been noticed that the best performance is achieved when this parameter is slightly less than the maximum possible number 1024, e.g., 960);

streams_CUDA is used to specify the number of random-number streams per one task on a CUDA device. The total number of streams per one device is obtained by multiplying streams_CUDA and tasks_CUDA. The program calculates the number of streams on the CPU by subtracting the numbers of streams on all used CUDA devices from the total number of streams (defined by the keyword NSTREAMS). If the resulting number is less than the number of CPU threads (defined by the keyword TASKS), then a corresponding error message is displayed;

blockSize_CUDA is the number of CUDA threads per one block;

time_CUDA is used to specify the duration of a single invocation of the CUDA kernel (in milliseconds). The recommended value is 100 or less. If this number is too large, the computer may become unresponsive. If this happens, then after a few seconds the Nvidia video driver will usually recover (terminating MCNelectron_CUDA in the process), however, occasionally the recovery fails and the computer needs to be restarted. In addition, if the video card that is connected with the computer monitor is among the CUDA devices used for the simulation, then a too large value of that parameter may cause "stuttering" of the video signal, which may interfere with other work that is being done on the same computer at the same time. On the other hand, if this parameter is too small (e.g., less than 1), then the number of kernel calls may be become so large that the "overhead" caused by multiple kernel calls may become important and performance may drop;

ratio_CUDA is used to specify the fraction of particle histories that should be simulated by CUDA devices. If more than one CUDA device is used, then those histories are distributed equally among all used CUDA devices. During the simulation, the CPU and GPU workloads may be dynamically adjusted (see descriptions of the keywords "heap_CUDA" and "balance_CUDA" below);

heap_CUDA is used to specify how CUDA tasks start a new history. If that parameter is zero, then each task is assigned a fixed number of source particles (i.e., a fixed number of histories). This option precludes workload balancing between CUDA devices and the CPU; it should be used if exact repeatability of simulation results is needed. If that parameter is 1, then, after finishing a history, each task starts a new history if the total number of started histories on the CUDA device is less than the assigned total number of histories that have to be simulated; otherwise, the task becomes inactive. Thus, if histories that were initially assigned to a particular task were short, then that task may have enough time left for simulating extra histories and thus decreasing the workload of the tasks that are "lagging". Since the processing times of different histories are affected by various random factors, and since different tasks use different streams of random numbers, this option introduces a random component into the simulation results, i.e., they are no longer exactly repeatable (however, this is usually not a problem, because the aim of Monte Carlo simulations is estimation of statistical averages). If that parameter is 2, then each task does not follow a history from start to finish; instead, a "global" buffer of banked particles (consisting of secondary particles in all unfinished histories) is available to all tasks on a given CUDA device. After a particle is lost (e.g., due to escape from the layer or due to decrease of its energy below the cutoff value), the task first attempts to read the banked particle data from the mentioned buffer. If the read attempt is successful, then that particle is tracked (even if it belongs to a history that was started by another task). If the read attempt fails (i.e., if the mentioned buffer is empty) and if the total number of started histories on the CUDA device is less than the assigned total number of histories that have to be simulated, then the task generates a new source particle and starts a new history. The option "heap_CUDA 2" introduces an even larger random component into simulation results than the option "heap_CUDA 1" does (however, this does not affect the statistical averages). In the case "heap_CUDA 2", the number of source particles is allowed to be less than the number of tasks. ***Notes***: **1)** Pulse-height tallies can be calculated only with "heap_CUDA 0" or "heap_CUDA 1". **2)** Workload balancing between CUDA devices and the CPU is possible only with "heap_CUDA 1" or "heap_CUDA 2" (see the description of the keyword "balance_CUDA" below);

balance_CUDA is a "switch" that "turns on" workload balancing between CUDA devices and the CPU. When this feature is "turned on", then during the simulation the number of histories assigned to the CPU and to each CUDA device is dynamically adjusted on the basis of performance of the CPU and of each CUDA device, in an attempt to minimize differences between total processing times of the CPU and of each CUDA device. The workload balancing in conjunction with option "heap_CUDA 1" or "heap_CUDA 2" eliminates the need to determine the optimal value of the parameter "ratio_CUDA" beforehand (the latter parameter defines only the initial distribution of workload, which is automatically adjusted during the simulation). ***Notes***: **1)** Workload balancing can not be used together with the option "heap_CUDA 0". **2)** When workload balancing is turned on, MCNelectron_CUDA may occasionally simulate up to four "extra" histories;

avgBanked_CUDA is used to specify the memory amount that must be allocated on a CUDA device for storing the banked particle data (the "banked particles buffer"). If this parameter is positive, then it is interpreted as the average number of banked particles per one task when the mentioned buffer is full. If this parameter is negative, then it is interpreted as the fraction of the global memory on each CUDA device that must be allocated for banked particles. A cautionary note regarding SLI, quoted from "CUDA C Programming Guide" [2]: "an allocation in one CUDA device on one GPU will consume memory on other GPUs that are part of the SLI configuration of the Direct3D or OpenGL device. Because of this, allocations may fail earlier than otherwise expected." Thus, if there are two Nvidia video cards in SLI configuration, then values of the allocated memory fraction close to 0.5 or greater (for example, "avgBanked_CUDA -0.6") will

cause a CUDA memory error (except for video cards with more than 4 GB of video memory). If there are more than two cards in SLI configuration, then the memory error may occur at an even lower fraction of the memory specified. For this reason, if a large number of banked particles are expected, it is recommended to disable SLI before the simulation.

maxBankedRatio_CUDA is used to specify the size of the "index buffer", i.e., the memory amount that must be allocated per one task for storing positions of banked particles in the above-mentioned banked particles buffer. The "index buffer" is used only with "heap_CUDA 0" and "heap_CUDA 1", because in those cases each task tracks only those secondary particles that were created by the same task in the same history. The number that follows this keyword is interpreted as the ratio of the number of elements of "index buffer" to the average number of banked particles per one task (defined by the keyword "avgBanked_CUDA");

bankModeThr_CUDA controls the "bank mode" when option "heap_CUDA 2" is used. The term "bank mode" refers to the choice of an electron that has to be banked after an impact ionization event: the banked electron may be either the lower-energy one (the so-called "secondary" or "knock-on" electron), or the higher-energy one (the "primary" electron). Normally, when option "heap_CUDA 2" is used, the knock-on electron is the one that is banked. However, this may cause an overflow of the banked particles buffer in the case of high energies of source particles and thick targets. Consequently, the bank mode is changed when the filling fraction of the banked particles buffer exceeds a certain value. That "threshold" value of the filling fraction is specified after this keyword. Its default value is 0.8 (i.e., while the banked particles buffer is less than 80 % full, the knock-on electron will be banked, and when the banked particles buffer is more than 80 % full, the primary electron will be banked). ***Note***: In the case "heap_CUDA 0" or "heap_CUDA 1", this keyword is ignored (in those cases, the primary electron is always the one that is banked);

first_CUDA is used to specify the sequence number of the first CUDA device in the range of CUDA devices used for the simulation (CUDA devices are numbered sequentially starting from 1);

last_CUDA is used to specify the sequence number of the last CUDA device in the range of CUDA devices used for the simulation (CUDA devices are numbered sequentially starting from 1). If firstCUDA is 1 and last_CUDA is 0, then all available CUDA devices will be used;

use_CUDA is used to specify the sequence numbers of the CUDA devices that must be used for the simulation. When specified on the command line, those numbers must be comma-delimited. For example, "use_CUDA 1,3" means that CUDA devices No. 1 and No. 3 have to be used. ***Notes***: **1)** When specified in the input file, the numbers may be either comma-delimited or space-delimited. **2)** The keyword "use_CUDA" can not be used together with keywords "first_CUDA", "last_CUDA", or "skip_CUDA". **3)** If the keyword "use_CUDA" is missing, then the range of used CUDA devices will be determined according to the numbers entered after the above-mentioned keywords "first_CUDA" and "last_CUDA".

r_CUDA is used to specify the workload ratios for each of the CUDA devices that must be used for the simulation. I.e., the program divides the total CUDA workload ratio (specified using the mentioned keyword "ratio_CUDA") among the CUDA devices proportionally to the numbers specified after this keyword. When specified on the command line, those numbers must be comma-delimited. This keyword can be used only together with the mentioned keyword "use_CUDA", and the number of entries after "r_CUDA" must be the same as the number of entries after "use_CUDA". For example, in the case "use_CUDA 1,3 r_CUDA 2,1 ratio_CUDA 0.6" the workload of the CUDA device No. 1 will be twice larger than the workload of the CUDA device No. 3. Since in this example the total workload of the CUDA devices is 60 %, the CUDA device No. 1 will simulate 40 % of all histories, and the CUDA device No. 3 will simulate 20 % of all histories. ***Notes***: **1)** When specified in the input file, the numbers may be either comma-delimited or space-delimited. **2)** If any of the workload ratios specified after "r_CUDA" is zero, the corresponding CUDA device will not be used. **3)** If the keyword "r_CUDA" is missing, the workload will be divided equally among all used CUDA devices.

skip_CUDA is used to specify the sequence numbers of the CUDA devices that must be skipped, i.e., not used for the simulation. When specified on the command line, those numbers must be comma-delimited. For example, "skip_CUDA 2,4 first_CUDA 1 last_CUDA 6" is equivalent to "use_CUDA 1,3,5,6". The keyword "skip_CUDA" can not be used together with "use_CUDA". ***Note***: **1)** When specified in the input file, the device numbers listed after "skip_CUDA" may be either comma-delimited or space-delimited. **2)** If the keyword "skip_CUDA" is missing, then there will be no skipped CUDA devices, i.e., the sequence numbers of the used CUDA devices will be entirely defined using the above-mentioned keywords "first_CUDA" and "last_CUDA", or using the list of devices specified after the above-mentioned keyword "use_CUDA".

stats_CUDA controls output of CUDA device statistics to files. The allowed values are:

0 – no CUDA device statistics will be written to files,

1 – the summary statistics will be included in the output file,

2 – second-by-second statistics will be written to files in the same subfolder where the tally data are (the subfolder name format is "Files_<output_file_name>"). The names of files with second-by-second statistics have format "d<n>.txt", where "<n>" is the sequence number of a CUDA device.

3 – both the summary statistics and second-by-second statistics will be written.

[For more information about CUDA device statistics, see Section 4.7.]

The remaining 4 CUDA-specific keywords control "emulation" of CUDA devices by the CPU (those keywords may be used only on the command line). Here, the term "emulation" means that the particle histories are simulated by the CPU using exactly the same random number streams that would be used by the CUDA device. I.e., the only difference between a normal CPU-only computation and a computation done in emulation mode is that the random number streams are assigned to particle histories differently. In emulation mode, assignment of random number streams to particle histories is controlled by the mentioned keywords "NSTREAMS", "tasks_CUDA", "streams_CUDA", "ratio_CUDA" and "r_CUDA". This makes it possible to reproduce the results of a hybrid CPU/GPU simulation using only the CPU or using only a part of available CUDA devices. For exact reproduction of the results in emulation mode, the options "rand_CUDA 1" and "heap_CUDA 0" are required. Below are descriptions of the four emulation-related keywords:

em_CUDA is used to specify the sequence numbers of the CUDA devices that must be emulated. Those numbers must be comma-delimited. For example, "em_CUDA 1,3" means that CUDA devices No. 1 and No. 3 have to be emulated. ***Notes***: The keyword "use_CUDA" can not be used together with keywords "emFirst_CUDA", "emLast_CUDA", or "emulate_CUDA", which are described below;

emFirst_CUDA is used to specify the sequence number of the first emulated CUDA device in the range of emulated CUDA devices (CUDA devices are numbered sequentially starting from 1);

emLast_CUDA is used to specify the sequence number of the last emulated CUDA device in the range of emulated CUDA devices (CUDA devices are numbered sequentially starting from 1);

emulate_CUDA is a "switch" that "turns on" emulation of all devices specified after the keyword "use_CUDA" or belonging to the range of devices defined using the keywords "first_CUDA" and "last_CUDA". This option can not be used together with "em_CUDA", "emFirst_CUDA" or "emLast_CUDA".

The default values of the CUDA-specific parameters are the following:

rand_CUDA	2
seed_CUDA	123
tasks_CUDA	0
threads_CUDA	960
streams_CUDA	1

blockSize_CUDA	8
time_CUDA	100
ratio_CUDA	0.5
heap_CUDA	1 or 2 (depending on whether pulse-height tallies are used or not)
balance_CUDA	1
avgBanked_CUDA	-0.2
maxBankedRatio_CUDA	3
bankModeThr_CUDA	0.8
first_CUDA	1
last_CUDA	0
stats_CUDA	0
emFirst_CUDA	0
emLast_CUDA	0
emulate_CUDA	0

4.7. CUDA device statistics

MCNelectron_CUDA outputs CUDA device statistics in three ways:

1) during the simulation, the second-by-second statistics as well as current values of the summary statistics are displayed on one line in the console window (that line is updated every second);
2) during the simulation, second-by-second statistics may be optionally written to a file (as explained in Section 4.6, this is achieved using the option "stats_CUDA 2" or "stats_CUDA 3");
3) after the simulation is finished, the summary statistics may be optionally written to the output file (as explained in Section 4.6, this is achieved using the option "stats_CUDA 1" or "stats_CUDA 3").

The formats of each of those three outputs are described below.

One-line statistics in the console window

Since all statistics can not fit on a single line, MCNelectron displays only a portion of the total set of statistics at a time. The displayed set of statistics is selected by repeatedly pressing the "+" or "-" key on the keyboard during the simulation (the key "+" cycles the statistics forward, and the key "-" cycles the statistics backwards). There are three lines of statistics for each CUDA device used and one line with some overall statistics. The latter line also contains the statistics pertaining to the part of the simulation that is done by the CPU. If two or more CUDA devices are used, then it is possible to cycle between statistics of the same kind for different devices (the CUDA devices are numbered sequentially starting from 1, and the CPU is the device No. 0). This is achieved by repeatedly pressing the "]" or "[" key on the keyboard during the simulation. After pressing the key "]", the same type of statistics for the next device will be displayed, and after pressing the key "[", the same type of statistics for the previous device will be displayed (if the next or previous device is the CPU, then after pressing "]" or "[" the line with overall statistics will be displayed).

During a hybrid CPU/GPU simulation, the line with overall statistics could look like this:

```
3.160 s: 8883/30000 (29.61%), CPU - 423/19500 (2.17%), CUDA - 8460/10500 (80.57%), threads 1920/1920, time 99.02%, 99.94%.
```

This line has the following components:

- The time elapsed since the start of the simulation (in seconds).
- The total number of finished histories, the total number of source particles and their ratio. ***Note***: In the case "heap_CUDA 2", the number of finished histories on CUDA devices is replaced by the number of finished source tracks (see the description of output parameter "src" below).
- The number of finished histories on the CPU, the total number of histories assigned to the CPU, and their ratio.

- The number of finished histories on CUDA devices, the total number of histories assigned to the CUDA devices, and their ratio (the same note as above applies).
- The number of active CUDA threads and the total number of CUDA threads.
- Ratio of the total kernel execution time to the total elapsed time (both those times are calculated using the system timer). This ratio indicates the amount of "overhead" caused by multiple invocations of the kernel (because of that overhead, this ratio is less than 1).
- Ratio of the total useful time spent by all CUDA threads to the total execution time of all CUDA threads (both those times are calculated using the CUDA native high-performance timer). The "useful" time is the time when a thread is active (i.e., the time interval between starting the thread and terminating it). This ratio indicates differences between workloads of different CUDA threads on a CUDA device, as well as between workloads of different CUDA devices (because of those differences, that ratio is less than 1).

If the simulation is performed using two or more CUDA devices and not using the CPU, then in addition to the above-mentioned overall value of the useful-to-total time ratio, its values for each of the devices are given between parentheses (this component is not present in the above example).

During a CPU-only simulation, this line contains some statistics for the first four simulation threads, for example:

```
2.001 s: 2780/40000 (6.95%), active threads - 1:2220:2220 (4), 2:2844:2844 (4),
3:2845:2845 (1), 4:3035:3035 (1), ...
```

In this case, the following three colon-delimited numbers are given for each of the first four active threads: the thread number, the sequence number of the currently used random-number stream (if this number is equal to the thread number, then it is not shown), and the sequence number of the current history (in the total number of source particles). After those three numbers, the current number of banked particles in that thread is given in parentheses.

If control tallies are used (see Section 4.4), then the largest current relative error for those tallies is included at the end of the same line (the message format is explained in Section 4.4). In this case, some of the previous information is omitted (in order to keep this line sufficiently short).

The three mentioned lines with CUDA device statistics could look, for example, like this:

```
2.049 s: dev. 1-1 - src 912 / 1960 = 46.53% (199) rem 70939 (48943) tb 100.16
(100.16) tcy 164.74 (163.50) tcl 92.68 (97.94)

4.218 s: dev. 1-2 - nk 42 (11) brk 0.000% (0.000%) cyc 617.922 (633.486) nTh
5.544 (5.221) maxLoad 10.285% (6.117%) nw 0.00% (0.00%)

5.219 s: dev. 1-3 - nB 105555, 0.310% (402, 0.001%) nThR 191.774 (924.000) nThW
608.208 (294.727) nThN 1120.019 (701.273)
```

Each of those three lines starts with the time elapsed since the start of the simulation (in seconds), followed by the device number and the sequence number of the set of statistics for that device. After that, there are several numbers (or pairs of numbers) which have the meaning of certain sums or averages, calculated by summing or averaging certain quantities over all kernel calls for the current device (there is one exception: "nB" has the meaning of a maximum value, rather than a sum). Each of those numbers (or a pair of numbers) is preceded by its notation and followed by a number (or two numbers) in parentheses. The numbers in parentheses indicate the change of the corresponding sum, or the value of the corresponding average, during the last second. The first set of CUDA statistics consists of the following components:

src – the number of finished histories, the total number of histories (i.e., source particles) assigned to that device, and their ratio. In the case "heap_CUDA 2", the number of finished histories is replaced by the number of finished source tracks. A "source track" is a connected series of line segments, each one representing a free flight of a particle. A source track starts at the point of entry of the source particle and ends at the point where the last particle of a chosen

"path" in a given history is lost. If there is a "fork" in this "path" due to creation of a secondary particle, then the "branch" that is assigned to the source track is chosen according to arbitrary criteria (for example, if the source particle is a high-energy photon, then after a pair production event the source track will be continued by the electron created in that event, whereas the positron will be banked). Thus, a single source track may represent several particles. In a task that started a new history, the first attempt to read banked particle data after generating the source particle occurs immediately after the source track is finished. Since each source track starts with a source particle, the number of finished source tracks is less than or equal to the number of started histories and greater than or equal to the number of finished histories. The number of histories or source tracks finished during the last second is given in parentheses;

rem – the number of removed or replaced particles (or the sum of their weights if interaction forcing is used);

tb – average duration of a single block of CUDA threads during one kernel call (in milliseconds);

tcy – average duration of a single loop cycle in the kernel (in microseconds). A single loop cycle in one CUDA thread corresponds to a single call to a "state handler" mentioned in Section 1.3. In addition, a single loop cycle includes re-distribution of tasks among threads (see Section 1.3);

tcl – average duration of a single call to a state handler (in nanoseconds).

The second set of CUDA statistics consists of the following components:

nk – number of kernel calls;

brk – fraction of block calls ended by breaking the loop (as opposed to ending the loop because the assigned time is exceeded). This occurs when it is determined that all threads of the current block are inactive;

cyc – average number of loop cycles per one block call;

nTh – average number of active CUDA threads per one loop cycle;

maxLoad – fraction of loop cycles with maximum load (i.e., with all threads in the block active);

nw – fraction of loop cycles when a task, which was in the "read" state at the start of a cycle, remained in the "read" state at the end of that loop cycle (i.e., an attempt to read particle data from the banked particles buffer was unsuccessful and no more particle histories remained to be started). This can happen only with the option "heap_CUDA 2". Consequently, in the case "heap_CUDA 0" or "heap_CUDA 1" this fraction is always 0.

The third set of CUDA statistics consists of the following components:

nB – the maximum number of banked particles and its ratio to the maximum allowed number of banked particles. In the case "heap_CUDA 0" or "heap_CUDA 1", the maximum is per one task, and in the case "heap_CUDA 2", it is the maximum number of banked particles in the global buffer (the numbers in parentheses are calculated for the last kernel call).

nThR – average number of tasks that were in the "read" state at the start of a kernel call (a task is in the "read" state just before attempting to read banked particle data, or when it is inactive after finishing simulations of all histories that were assigned to it);

nThW – average number of tasks that were in the "write" state at the start of a kernel call (a task is in the "write" state just before simulating an interaction event characterized by creation of new particles, which may need to be banked);

nThN – average number of tasks that were in the "neutral" state at the start of a kernel call. A state is called "neutral" if it does not involve either reading from the banked particles buffer or writing to it (e.g., calculation of interaction cross sections).

Second-by-second CUDA device statistics written to files

The first line in the file with second-by-second CUDA device statistics contains the column headers, and the other lines contain the values of various statistics for each second of the simulation. The first column of that file contains values of the time elapsed since the start of the simulation (in seconds), rounded to the nearest second. The next 16 columns contain 15 previously-described statistics, calculated for each one-second interval and listed in the same order as in one-line statistics (see above), and one additional statistic "tw" (column No. 13), whose meaning is similar to the mentioned quantity "nw", with the only difference that "tw" is the fraction of time rather than the fraction of loop cycles. I.e., "tw" is the fraction of total time spent for unsuccessful reads from the banked particles buffer by a task that was in the "read" state at the start of a loop cycle, with a condition that no more source particles had to be generated. The headers of those columns are the same as previously-mentioned notations. The numbers in columns No. 2 – 12 ("src" to "nw") and No. 15 – 17 ("nThR", "nThW" and "nThN") are the same ones that would be displayed in parentheses in the one-line statistics during the simulation at the corresponding moment of time. In the case "heap_CUDA 0" or "heap_CUDA 1", the numbers in column No. 14 ("nB") have the meaning of the maximum number of banked particles per one history observed up to that moment in time, and in the case "heap_CUDA 2" they have the meaning of the maximum total number of banked particles observed during the last kernel call.

The next 18 columns (i.e., columns 18 to 35) contain fractions of the numbers of calls to various state handlers in the total number of calls. In other words, those are fractions of the total number of loop cycles spent for processing different stages of the simulation. The meanings of those 18 columns are explained below:

nPsig – calculation of photon interaction cross sections;
nPdist – calculation of the distance to photon collision;
nPinter – sampling of chemical element and interaction type in the case of a photon collision;
nCoh – simulation of coherent scattering of photons;
nIncoh – simulation of incoherent scattering of photons;
nPE – simulation of photoelectric absorption;
nPP – simulation of pair production;
nEsig – calculation of electron interaction cross sections;
nEdist – calculation of the distance to electron collision;
nEinter – sampling of chemical element and interaction type in the case of an electron collision;
nElast – simulation of elastic scattering of electrons;
nBrems – simulation of bremsstrahlung;
nExc – simulation of atomic excitation by electron collisions;
nIon – simulation of electron impact ionization;
nRelax – simulation of atomic relaxation;
nAnnih – simulation of positron annihilation;
nSource – generation of source particles;
nRead – reading banked particle data when a task in the "read" state at the start of a loop cycle.

The next 18 columns (i.e., columns 36 to 53) contain the time fractions corresponding to fractions of the number of calls given in the previous 18 columns. I.e., the only difference is that they give the fraction of simulation time spent for various states, instead of the fraction of the number of calls to various state handlers. The headers of those columns start with "t" instead of "n" (i.e., "tPsig", "tPdist", etc.).

The last two columns (No. 54 and No. 55) contain the time fractions spent for selecting the tasks to be processed in the current loop cycle (the corresponding column title is "tCheck1") and for updating numbers of tasks corresponding to various states after exiting from a state handler (the corresponding column title is "tCheck2"). The former of the two mentioned steps is done at the start of a loop cycle by the first thread of the block, and the latter step is done at the end of a loop cycle

by each thread of the block (in this step, multiple threads modify a number stored in one memory location, and such memory accesses are done serially).

If one of the options "stats_CUDA 2" or "stats_CUDA 3" has been specified in the input file or on the command line, then output of second-by-second CUDA device statistics to files can be stopped or resumed at any time during the simulation by pressing the 'S' or 'F' key on the keyboard, respectively.

Summary CUDA device statistics in output files

If one of the options "stats_CUDA 1" or "stats_CUDA 3" has been used, then average values of most of the time fractions mentioned above will be written to the output file. Several other statistics will be output as well, e.g., the maximum number of banked particles for each CUDA device and for the CPU. ***Note***: In the case "heap_CUDA 2", the maximum value of the *total* number of banked particles on each CUDA device will be output, whereas in the case "heap_CUDA 0" and "heap_CUDA 1" the maximum number of banked particles *in one history* will be output.

Since most CUDA statistics written to the output file have descriptive labels explaining their meaning, a separate explanation is not needed here. For an explanation of other information included in the output file, see Section 6.

5. Simulation of physical effects that are not included in specification of the ENDF/B library

In addition to various types of interactions and effects whose cross sections and probabilities are stored in the ENDF/B data library [5], MCNelectron simulates six physical effects, for which the ENDF/B-VI.8 library provides neither tabular probabilities nor an analytic prescription:

(a) non-isotropic angular distribution of bremsstrahlung photons,
(b) partial suppression of positron bremsstrahlung in comparison with electron bremsstrahlung,
(c) non-isotropic angular distribution of photoelectrons,
(d) non-uniform distribution of positron and electron energies during pair production,
(e) angular distribution of electrons and positrons created in pair production events,
(f) Doppler broadening of energy distribution of incoherently scattered photons.

For each of those effects, there is a keyword in the MCNelectron input file, which allows "turning on" or "turning off" simulation of the effect. By default, all those effects are simulated. If any of those effects is "turned off", then it will be replaced by a simplified model (e.g., isotropic bremsstrahlung) or neglected completely. The mentioned keywords, listed in the same order as the corresponding effects, are the following:

(a) BREMS_ANGULAR_DISTR,
(b) BREMS_POSITRON_CORRECTION,
(c) PE_ANGULAR_DISTR,
(d) PP_ENERGY_DISTR,
(e) PP_ANGULAR_DISTR,
(f) INCOH_DOPPLER.

For some of those effects, there are other keywords controlling some details of the simulation. Those keywords are explained in the descriptions of the methods used for simulating each of the mentioned effects, which are provided below.

(a) Non-isotropic angular distribution of bremsstrahlung photons

If the integer number that follows the keyword "BREMS_ANGULAR_DISTR" is non-zero, then bremsstrahlung photons will be emitted non-isotropically, with angular probability density depending both on incident electron energy and on emitted photon energy (this is the default behavior). Otherwise, the bremsstrahlung photons will be emitted isotropically. In any case, the

electron's direction is unchanged by the bremsstrahlung event. Simulation of bremsstrahlung angular distribution requires pre-computed values of angular probability density of bremsstrahlung. Those values are in the file "Brems_angular_prob_dens.dat", which is in subfolder "Data" of the MCNelectron distribution package. The probability density values that are stored in that file were calculated by integrating analytically over the triply differential cross-sections derived by Bethe and Heitler. Those calculations were done using a code that is a supplement to the article [6] (that code was downloaded from the website of the journal where that article was published). The data in the file "Brems_angular_prob_dens.dat" are in binary format. Below is a description of that format (nested sequences are indicated by indentations; the lines starting with the words "Sequence" and "End of sequence" are only included for clarity and are not represented in the data):

First 400 bytes of the file contain the starting positions in this file for each of 100 chemical elements,
Sequence 1, which is repeated 100 times (once for each chemical element):
    4 bytes contain the number of incident electron energies ("nInc"),
    Sequence 2, which is repeated nInc times:
        8 bytes contain the incident electron energy (eV),
        4 bytes contain the number of bremsstrahlung photon energies ("nBrem"),
        Sequence 3, which is repeated nBrem times:
            8 bytes contain the bremsstrahlung photon energy (eV),
            4 bytes contain the number of probability density values ("nProb"),
            Sequence 4, which is repeated nProb times:
                8 bytes contain the value of the cosine of the angular deflection of the photon,
                8 bytes contain the corresponding value of the probability density.
            End of sequence 4
        End of sequence 3
    End of sequence 2
End of sequence 1

The MCNelectron package includes the executable "Data\ExtractBremsAngularDistr.exe", which extracts ASCII data from the mentioned binary file and creates a subfolder "Brems_angular_prob_dens" with 100 human-readable files (a separate file for each chemical element). The meaning and order of numbers in each of those text files is the same as in a single "Sequence 1" in the above description of the binary format. Those text files can be used as input data for the simulation, too, instead of the mentioned binary file. This is controlled by the keyword "BREMS_BINARY_DATA". If the integer number that follows that keyword is non-zero, then the data will be loaded from the mentioned binary file "Data\Brems_angular_prob_dens.dat" (this is the default behavior). Otherwise, the data will be loaded from the mentioned ASCII files. In the latter case, the user has an option to specify the folder with those text files. The folder name must be specified after the keyword "BREMS_ANGULAR_DISTR_DIR" in the input file. The default name of the folder with text files containing angular probability densities of bremsstrahlung is "C:\Brems_angular_prob_dens\".

In the case of simulation of non-isotropic bremsstrahlung, the mentioned additional probability density data are only used when the incident electron energy (E) belongs to a pre-defined energy interval [E_{min}, E_{max}]. When $E < E_{min}$ or $E > E_{max}$, then a simple analytical probability distribution

$$p(\mu)\,\mathrm{d}\mu = \tfrac{1}{2}\,(1-\beta^2)\,/\,(1-\beta\,\mu)^2\,\mathrm{d}\mu \qquad (5.1)$$

is used, where β is the ratio of the electron speed and the speed of light, and μ is the cosine of the angular deflection of the photon relative to velocity vector of the incident electron or positron. The default values of E_{min} and E_{max} are 1 keV and 1 GeV, respectively. Due to simplicity of the function $p(\mu)$ defined by Eq. (5.1), the values of μ corresponding to that distribution are sampled analytically, by solving the following equation:

$$\int_{-1}^{\mu} p(\mu')\mathrm{d}\mu' = r\,, \qquad (5.2)$$

where r is a random number that is uniformly distributed between 0 and 1. By substituting the expression (5.1) into the integral (5.2) and solving for μ, the following expression of the sampled value of μ is obtained:

$$\mu = \frac{2r + \beta - 1}{\beta(2r-1)+1}. \tag{5.3}$$

The mentioned default value of $E_{\min}$ (i.e., 1 keV) is mainly for comparison with MCNP6: as stated in [1], this analytical distribution "is not really appropriate for low energies, and its presence in the current code is a temporary expedient". The user may set other values of $E_{\min}$ and $E_{\max}$. $E_{\min}$ must be specified in the input file after the keyword "BREMS_ANGULAR_DISTR_LOW_EN", and $E_{\max}$ must be specified after the keyword "BREMS_ANGULAR_DISTR_HIGH_EN" (the unit of energy is the same as for all other energies specified in the input file, i.e., it is defined by the keyword "E_UNIT" described n Section 4.1). In order to use a more realistic angular distribution of bremsstrahlung photons at low energies, $E_{\min}$ should be set to zero. If necessary, the analytical distribution defined by Eq. (5.1) can be applied for all values of electron energy. This is achieved by setting $E_{\min} = E_{\max}$. Then the values of $E_{\min}$ and $E_{\max}$ are not used during simulation, and the data files with bremsstrahlung angular probability density data are not loaded (in such a case, the text entered after the keywords "BREMS_BINARY_DATA" and "BREMS_ANGULAR_DISTR_DIR" is ignored).

(b) Partial suppression of positron bremsstrahlung in comparison with electron bremsstrahlung

As reported in [7], the ratio of the radiative stopping powers for positrons and electrons can be approximated as a function only of the variable E / Z^2, where E is the energy of the incident electron or positron and Z is the atomic number. The values of that ratio, which are tabulated in [7], can be approximated by the following function of the variable $x = \ln(E / Z^2) - 5.87522$, where E is expressed in units of eV:

$$\frac{\Phi^+_{\text{rad}}}{\Phi^-_{\text{rad}}} = \begin{cases} 0.020 & (E/Z^2 \le 0.1), \\ 0.606133 + 0.140534x + 0.0102012x^2 + 0.000217602x^3 & (\ln(0.1) - 5.87522 < x \le 0), \\ 0.606133 + 0.140534x - 0.0163087x^2 + 0.000609304x^3 & (0 < x < \ln(5\cdot 10^5) - 5.87522), \\ 1 & (E/Z^2 \ge 5\cdot 10^5). \end{cases} \tag{5.4}$$

The relative deviation of the values calculated according to (5.4) from the values tabulated in [7] is less than 1.4 % for all values of E.

If the current particle is a positron, then the mentioned partial suppression of positron bremsstrahlung is taken into account by multiplying the electron bremsstrahlung cross section by the factor calculated according to Eq. (5.4). This is the default behavior. The user may "turn off" simulation of this effect using the keyword BREMS_POSITRON_CORRECTION in the MCNelectron input file. If the number that follows that keyword is zero, then this effect will not be simulated, i.e., the positron bremsstrahlung cross section will be equal to the electron bremsstrahlung cross section.

(c) Non-isotropic angular distribution of photoelectrons

If the integer number that follows the keyword "PE_ANGULAR_DISTR" is non-zero, then photoelectrons will be emitted non-isotropically (this is the default behavior). Otherwise, the photoelectrons will be emitted isotropically. In the case of non-isotropic emission of photoelectrons, two forms of their angular distribution are used, depending on the energy of the emitted electron. For electron energies below 50 keV, the angular probability density function $p(\mu)$ is of the form

$$p(\mu)\mathrm{d}\mu \sim (1-\mu^2)[\mathrm{Im}((\alpha + \mathrm{i}\cdot s)^{-n'-1}) / s]^2 \,\mathrm{d}\mu, \tag{5.5}$$

where μ is the cosine of the angle between incident photon and photoelectron wave vectors, s is the absolute value of the vector that is equal to the difference of those two wave vectors, "i" is the

imaginary unit, and notation "Im(…)" means the imaginary part of a complex number. If the electron is ejected from one of the first 4 shells (K, L, M or N), then n' coincides with the principal quantum number (n) of that shell. For higher shells, n' is fixed at 4. α is the factor in the exponent of the approximate (nodeless) wave function of the electrons in that shell. The approximate wave function is of the form

$$\psi(r) \sim r^{n'-1} \mathrm{e}^{-\alpha r}, \tag{5.6}$$

$$\alpha = \frac{Z - \sigma}{n^* a_\mathrm{B}}, \tag{5.7}$$

where Z is the atomic number, σ is the screening constant, n^* is the effective quantum number and a_B is the Bohr radius. The values of n^* and σ are calculated according to the Slater rules [8]. Eq. (5.5) has been obtained by calculating the matrix element of the perturbation operator describing the incident electromagnetic plane wave, when the initial state of the electron is described by the wave function (5.6) and the final state of the electron is described by another plane wave. In [9], this procedure was applied to the case when $n^* = n' = n = 1$ (K electrons). The angular distribution corresponding to the latter case is called "the Fischer distribution".

For photoelectron energies above 50 keV, the angular probability density function $p(\mu)$ is

$$p(\mu)\mathrm{d}\mu \sim \frac{1-\mu^2}{(1-\beta\mu)^4}\left[1 + \frac{1}{2}\gamma(\gamma^2-1)(1-\beta\mu)\right]\mathrm{d}\mu, \tag{5.8}$$

where γ is the ratio of the photoelectron kinetic energy and the electron rest energy, β is the ratio of the electron speed and the speed of light, and μ is the cosine of the angle between directions of the incident photon and the photoelectron. The distribution defined by Eq. (5.8) is called "the Sauter distribution" [10].

The mentioned value of 50 keV, which corresponds to transition from the low-energy angular probability density (5.5) to the high-energy angular probability density (5.8), is the same as in MCNP, where a similar (although probably not exactly the same) approach has been applied [1, 11]. This "transition energy" may be modified by the user. Its value must be specified in the input file after the keyword "PE_ANGULAR_DISTR_THR" (the unit of energy is the same as for all other energies specified in the input file, i.e., it is defined by the keyword "E_UNIT" described n Section 4.1).

In addition, it is possible to modify the number of values of μ in the tables of photoelectron angular probability densities, which are calculated by MCNelectron before the simulation. The default value of that number is 21 (i.e., μ is varied in increments of 0.1 from −1 to 1). In order to change that number, it must be specified in the input file after the keyword "PE_ANGULAR_DISTR_N". In any case, the values of μ will be equidistant.

(d) Non-uniform distribution of positron and electron energies during pair production

If the integer number that follows the keyword "PP_ENERGY_DISTR" is non-zero, then the energy distribution of positrons emitted when a photon with energy E is absorbed in a pair-production event is non-uniform (this is the default behavior). Otherwise, the positron energy will be sampled from a uniform distribution. Once the positron energy is sampled, the electron energy is uniquely determined by conservation of energy.

The non-uniform energy distribution function is calculated using the following formula, which is an approximation of the Bether-Heitler singly differential cross section for pair production:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}x} \sim [x^2 + (1-x)^2]\varphi_1 + \frac{2}{3}x(1-x)\varphi_2, \tag{5.9}$$

where $x = (E_+ + m_\mathrm{e}c^2) / E$ is the reduced energy, i.e., the fraction of the photon energy carried off by the positron (E_+ is the positron kinetic energy, m_e is the electron rest mass, c is the speed of light), and factors φ_1 and φ_2 are defined as follows:

$$\varphi_1 = 2[1 + \ln(a^2)] - 2\ln(1+b^2) - 4b\arctan(b^{-1}), \tag{5.10a}$$

$$\varphi_2 = 2[(2/3) + \ln(a^2)] - 2\ln(1+b^2) + 8b^2[1 - b\arctan(b^{-1}) - 0.75\ln(1+b^{-2})], \qquad (5.10b)$$

where a only depends on the atomic number Z:

$a = 122.8$ for $Z = 1$,
$a = 90.8$ for $Z = 2$,
$a = 100$ for $Z = 3$,
$a = 106$ for $Z = 4$,
$a = 111.7$ for $Z > 4$,

and b is defined as follows:

$$b = \frac{a \cdot Z^{-1/3} m_e c^2 E}{2(E_+ + m_e c^2)(E - E_+ - m_e c^2)}. \qquad (5.10c)$$

The expression (5.9) has been obtained from the formula provided in [10] by eliminating the terms proportional to the high-energy Coulomb correction (as stated in [10], "because of the approximate nature of this correction, it should not be used for photon energies of less than about 100 MeV"). The two terms of Eq. (5.9), which are proportional to φ_1 and φ_2, are the same as in a more complex formula derived in [12]. However, the expressions of the factors φ_1 and φ_2 that are given in [10] are slightly different from their definitions provided in [12]. The expressions (5.10a) and (5.10b) have been taken from [12].

The argument of the positron energy probability density that is used during simulation is defined slightly differently than the reduced energy x in Eq. (5.9): it is the fraction y of the total *kinetic* energy $T_{max} = E - 2m_e c^2$ that is carried away by the positron. Unlike x, the variable y can be equal to any number from 0 to 1. Before starting the simulation, MCNelectron calculates tables of positron energy probability densities $p(y)$ corresponding to 12 values of the incident photon energy from 1022 keV to 100 GeV. During simulation, the probability densities corresponding to the current photon energy are calculated by interpolation.

(e) Angular distribution of electrons and positrons created in pair production events

If the integer number that follows the keyword "PP_ANGULAR_DISTR" is non-zero, then the directions of positrons and electrons emitted in pair production events will be sampled using the probability density function $p(\mu)$ defined by Eq. (5.1), where β is the ratio of the particle speed (i.e., either positron speed or electron speed) and the speed of light [10]. The corresponding formula for calculating the sampled values of μ is (5.3). This is the default behavior. Otherwise (i.e., in the case "PP_ANGULAR_DISTR 0"), both the positron and the electron will continue in the direction of the incident photon.

(f) Doppler broadening of energy distribution of incoherently scattered photons

If the integer number that follows the keyword "INCOH_DOPPLER" is non-zero, then MCNelectron simulates the so-called Doppler broadening of the energy spectrum of incoherently scattered photons (caused by the distribution of the electron momentum inside the atom). This is the default behavior. Otherwise, the energies of the scattered photon and the recoil electron will be calculated as though the electron is initially free and at rest. Doppler energy broadening is simulated exactly as described in [13, 14]. This simulation requires the so-called Compton profiles for each subshell of each chemical element that is present in the target material. Those Compton profiles were published in [15]. They are stored in the ASCII file "Data\ComptonProfiles.dat", which is included in the MCNelectron distribution package. That file was downloaded from http://ftp.esrf.eu/pub/scisoft/xop2.3/DabaxFiles/ComptonProfiles.dat .

Another aspect of incoherent scattering of photons is controlled by the keyword "INCOH_SUBTRACT_BINDING_E". If the integer number following that keyword is non-zero, then the recoil electron energy is additionally reduced by subtracting the binding energy of the subshell from which the electron was ejected (if the resulting energy value is negative, then the

recoil electron is not emitted). This is the default behavior. Otherwise, the recoil electron energy is simply the difference of the incident and scattered photon energies. Thus, in the case with "INCOH_SUBTRACT_BINDING_E 0" and with simulation of Doppler energy broadening "turned on", conservation of energy and momentum is not simulated accurately (except in the high-energy limit, when atomic binding effects become negligible): the scattered photon energy is calculated under the assumption that the electron initially has some momentum, whereas the recoil electron energy is calculated under the assumption that its initial energy (and momentum) is zero. Another non-physical result of such simple calculation of the recoil electron energy is related to possible subsequent emission of X-ray photons or Auger electrons from the ionized atom: those secondary photons and electrons also carry off some energy and interact with the target material. Thus, the total energy carried off by the secondary particles (the recoil electron and the secondary photons and electrons from atomic relaxation) is slightly greater than the energy loss of the photon. This result violates conservation of energy. If the integer number following the keyword "INCOH_SUBTRACT_BINDING_E" is non-zero, then the violation of conservation of energy is eliminated: the mentioned binding energy is, in effect, the excitation energy of the ionized atom, and a part of that excitation energy may be carried off by Compton fluorescence photons and Auger electrons.

In any case, the direction of motion of the recoil electron is the same as the direction of the photon momentum transfer vector (i.e., the difference of the incident and scattered photon momentum vectors). This also is only accurate in the high-energy limit: since the mass of the target atom is much larger than the electron mass, the atom can absorb any amount of recoil momentum and, consequently, the directions of the scattered photon and the electron are not constrained by kinematics [10].

6. MCNelectron output file format

An MCNelectron or MCNelectron_CUDA output file consists of the following 7 sections:

- the command line that was used to generate the current file,
- contents of the input file and of the alternative cross sections info file (if it was used),
- the elapsed time and maximum number of banked particles,
- CUDA device statistics (optional; see Section 4.7),
- fraction of source photons absorbed or scattered and fraction of their energy absorbed,
- tables with summary statistics characterizing various types of energy-loss interactions,
- average energy loss per one secondary electron.

Information about energy-loss interactions is presented in six tables: three tables with information pertaining to creation and loss of photons and three tables with information pertaining to creation and loss of electrons and positrons. Each of those six tables consists of three columns: number of interaction events, their weight per source particle and the total energy of secondary particles created in those events or the total energy lost in those events (the energy is also given per source particle). All statistics under "electrons" include the contribution from positrons, too. When the source emits photons, there are two additional lines before the tables, with the values of the fraction of source photons that have been scattered or absorbed in the layer and the fraction of source energy that has been absorbed.

There are two tables pertaining to photon creation and two tables pertaining to electron creation: the first of those tables only contains statistics for the events when the energy of the created photon or electron exceeded the low-energy cutoff value specified in the input file (see Section 4.1, keywords CUT_P and CUT_E, respectively). The second of those tables contains statistics for all particle-creation events, including those when the energy of the created particle was less than the cutoff value. The first of those tables is mainly included for comparison with MCNP6 output, because, as mentioned in Section 1, MCNP6 only outputs statistics pertaining to particles with energy greater than the low-energy cutoff.

In order to facilitate comparison with MCNP6 output, the names of the rows in the mentioned tables and their order are the same as in the MCNP6 output file. There is only one item in MCNelectron output that is absent in the MCNP6 output: the table with particle and energy loss information has a row "backscatter", which contains the number and energy of particles that escaped from the layer through the surface that is exposed to the incident beam. The number and energy of photons or electrons that escaped from the other surface can be obtained by subtracting the values in the row "backscatter" from the corresponding values in the row "escape". Another difference between the tables with energy loss information in MCNelectron and MCNP6 output files is in the meaning of the number "N" (column "tracks" in MCNP6 output). In MCNP6 output, that number is the actual number of lost particles. Electrons and positrons can only be lost by three mechanisms:

- escape from the layer,
- decrease of electron energy below the low-energy cutoff value,
- positron annihilation (it occurs after the positron energy drops below the low-energy cutoff).

Photons can only be lost by four mechanisms:

- escape from the layer,
- decrease of photon energy below the low-energy cutoff value,
- photoelectric absorption ("capture"),
- pair production.

Consequently, the number of events given for all other energy loss mechanisms in the MCNP6 output file is zero. Conversely, MCNelectron outputs the number of *all* energy-loss events. Consequently, the number in the column "N" of the energy-loss table is non-zero whenever the lost energy is non-zero, even when that type of events did not cause loss of the incident particle (in such a case, only the energy loss values can be compared between MCNelectron and MCNP6 output). However, for comparison with MCNP6 output, the number in the row "Total" of the energy-loss tables in MCNelectron output files only includes the events when particles were actually lost.

The last line contains the average energy loss per one secondary electron (the so-called "*W*-value"). Two values of that energy loss are given: 1) taking into account only the electrons with energy greater than the low-energy cutoff value; 2) taking into account all secondary electrons, including those that were not tracked due to low-energy cutoff. Obviously, the latter value is closer to the true *W*-value of the target material.

7. Information about MCNelectron test files

The subfolder "Test" contains nine MCNelectron input files and nine MCNP6 input files corresponding to identical simulation conditions, together with respective output files. In addition, there is an MCNelectron input file where the option "ION_DWBA 1" is specified, and the corresponding output file (since this option is not available in MCNP, there are no corresponding MCNP6 input and output files). Input file names start with the word "input", and output file names start with the word "output". Five of the mentioned 9 simulations use a source of monoenergetic electrons, two simulations use a source of monoenergetic photons (the names of the corresponding MCNelectron input files end with "P"), one simulation uses a source of X-ray photons with continuous spectrum (the name of the corresponding MCNelectron input file ends with "X-rays"), and one simulation uses a source of monoenergetic positrons (the name of the corresponding MCNelectron input file ends with "E+"). The names of MCNP6 input and output files are obtained by appending "_MCNP6" to the names of corresponding MCNelectron input and output files.

Each of the 10 simulations using MCNelectron can be run either manually (by entering a command in the console window), or by executing one of the batch files, which are in the "Test" subfolder. The file "Batch_E.bat" runs the simulations where the source particles are electrons or positrons, and the file "Batch_P.bat" runs the simulations where the source particles are photons. The other two batch files ("Batch_E_MCNP6.bat" and "Batch_P_MCNP6.bat") attempt to run the corresponding simulations using MCNP6 in single-event mode.

A more detailed information about the test simulations is presented below.

In two of the test simulations, the main aim is estimation of the average energy per one secondary electron (the so called "*W*-value") when the target material completely absorbs energy of incident monoenergetic electrons. In both those simulations, the target material is pure xenon. Both those simulations make use of the ability of MCNelectron to "turn off" elastic scattering of electrons and tracking of particle coordinates (this makes the simulation much shorter). The corresponding MCNelectron input files are:

"input_Xe_1MeV_10.96_W-value_E.txt": energy of incident electrons is 1 MeV,
"input_Xe_100MeV_10.96_W-value_E.txt": energy of incident electrons is 100 MeV.

The number after the energy value in those two file names is the low-energy cutoff value for electrons (eV). It is equal to ionization energy of xenon (as stored in the ENDF/B library). In order to ensure that all energy of source electrons is absorbed by the target material when MCNP6 is used, the geometry of the simulated system is defined in the MCNP6 input files as an isotropic point source at the center of a sphere with radius 10^{12} cm, which is filled with xenon at a density of 1 g/cm^3.

In six test simulations, interaction of source particles with a layer composed of Xe, Ar and He with atomic fractions 0.1, 0.2, 0.7 is simulated (since the aim of those simulations is comparison of MCNelectron output with MCNP6 output, the composition of the target material has been chosen arbitrarily, without any regard to its practical or scientific value). Density of the target material is 1 g/cm^3 and thickness is 1 cm, excluding the simulation with the continuous-spectrum X-ray source, where the thickness is 10^{-5} cm. It should be noted that results of the simulation only depend on the value of mass thickness ρd, where ρ is density and d is thickness of the layer. Three of those six simulations use a monoenergetic source emitting particles with energy 1 MeV, and only differ by the particle type (electrons, photons or positrons). The fourth and fifth simulations use a monoenergetic source that emits photons or electrons with energy 100 MeV. Those two simulations are the most complex ones, because all physical processes and effects that MCNelectron can simulate are well represented in those two simulations. The sixth simulation uses a realistic spectrum of X-rays emitted by a Mo-tube with anode voltage 35 kV and filtered by a layer of Pyrex glass with thickness 1.2 mm. In order to make the definition of X-ray spectrum compatible with MCNP input format, each line of that definition ends with the character '&', which is used as a line-continuation character in MCNP input files (in MCNelectron input files, that character is not required and it is ignored by MCNelectron).

The last simulated system is a 25 keV electron beam that is incident normally on a 2 μm layer of U_3O_8 with density 8.3 g/cm^3. This is the only simulation using a plane-crossing tally (a table of energy distribution of photons emitted in the direction opposite the incident beam). There are two versions of that simulation: one of them uses the option "ION_DWBA 0", and another one uses "ION_DWBA 1". An MCNP6 counterpart exists only for the version with "ION_DWBA 0".

All test simulations were done using MCNelectron v1.1.1 or MCNelectron_CUDA v1.1.1 (that is why the tally data do not include the relative standard deviations, which were introduced in v1.1.2). The first 8 mentioned test simulations used 8 concurrent threads on the CPU, and the simulations with U_3O_8 used 10 CPU threads. In addition, MCNelectron_CUDA used two Nvidia GeForce GTX 780 Ti cards for the two simulations with U_3O_8. All simulations were done on a computer with an Intel Core i7-4930K processor running 64-bit Windows 8.1. In order to avoid system slowdown, it is recommended to set the number of CPU threads to a value that does not exceed the number of logical processors (the latter number can be seen in Windows Tasks Manager, tab "Performance"). In the case of simulations that use CUDA devices, the number of CPU threads specified after the keyword "TASKS" must be additionally decreased by 1, because, as mentioned in Section 4.1, that number does not take into account one CPU thread that is reserved for exchanging data with CUDA devices (that thread is not used for the actual simulation of particle interactions). ***Note***: If the number of threads does not exceed the number of logical processors on the computer, then the computer time specified in MCNP6 output files is larger than the true

simulation time by a factor equal to the square of the number of threads. In the case of 8 threads, the true time of an MCNP6 run is obtained by dividing the specified time by 64. In the case of 10 threads, the true time of an MCNP6 run is obtained by dividing the specified time by 100.

Comparison of MCNelectron and MCNP6 output data shows a very good agreement between the simulation results computed by those two programs. Comparison of the times of CPU-only simulations shows that MCNelectron significantly outperforms MCNP6 in all cases. In hybrid CPU/GPU computations using two GTX 780 Ti cards, the computation time is additionally reduced approximately by half.

8. Information about files in the subfolder "Simulations\X-rays" of the distribution package

The subfolder "Simulations\X-rays" of the MCNelectron distribution package contains 99 MCNelectron_CUDA input files and the corresponding output files (the output file name is obtained by appending "_out" to the input file name). Each of them is used to simulate characteristic X-ray emission from a thick target composed of one of those elements: C, Al, Si, Ti, Fe, Cu, Zn, Ge, Zr, Sn, Sm, Ta, W, Pt, Au and Pb, bombarded by monoenergetic electrons with energy from 5 keV to 30 keV (the element symbol and electron energy are included in the names of input and output files). Those simulations were performed using ENDF/B-VI.8 data, with the exception of inner-shell electron impact ionization cross sections, which were calculated using the distorted-wave Born approximation (this is achieved by specifying the option "ION_DWBA 1" in the MCNelectron input file). Interaction forcing was used (see Section 4.5). Those simulations were performed using MCNelectron v1.1.2 in "hybrid" CPU/GPU mode on a computer with an Intel Core i7-4930K processor and two Nvidia GeForce GTX 780 Ti video cards. The operating system was 64-bit Windows 8.1.

Two incidence angles were used: 0° and 45°. In the latter case, the file names contain the character string "45deg". The tallied photons are the ones whose energy corresponds to the K_α, K ($=K_\alpha + K_\beta$) or L_α characteristic X-ray line (the energy range containing that energy value is specified using the keyword "E1" in the input file). In the case of normal incidence, the tallied photons are either K_α (for the elements Ti, Cu and Ge) or K (for the elements C, Al and Si). In the case of non-normal incidence, the tallied photons are either K_α (for the elements Ti, Fe, Cu, Zn and Zr) or L_α (for the elements Sn, Sm, Ta, W, Pt, Au and Pb). In the latter case, the characteristic X-ray line is specified by the letter "K" or "L" inserted after "45deg_".

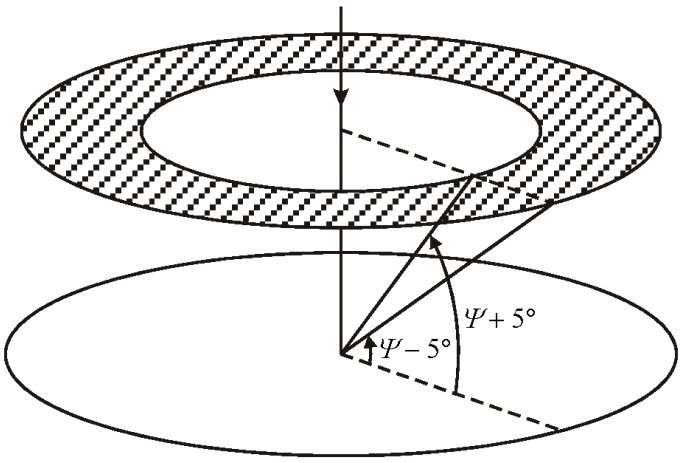

Fig. 1. Simulation geometry used to compute the X-ray yields

In all cases, an annular detector was assumed, i.e., only the photons hitting a ring-shaped area centered above the point of incidence were counted (see Fig. 1). The distance between the plane of incidence and the detector was 10000 cm, i.e., much larger than the part of the target where the photons originate. This, together with the fact that radiation emitted during atomic relaxation is isotropic, justifies the use of the annular detector even when the angle of incidence is not equal to 0°. The outer and inner radii of the ring correspond to takeoff angles $\Psi - 5°$ and $\Psi + 5°$, respectively, where $\Psi = 40°$ in the case of normal incidence, and $\Psi = 45°$ in the case of non-normal incidence. Each simulation was terminated when the relative standard deviation of the photon count corresponding to the mentioned characteristic X-ray line dropped below 2 %, or when the total number of source particles reached $5 \cdot 10^7$.

The computed tally data are in subfolders of the folder "X-rays". The names of those subfolders include the names of the corresponding output files. In order to obtain the characteristic

X-ray yield (in units of photons / (sr · electron)), the number of tallied photons must be divided by the solid angle subtended by the annular detector. The solid angle corresponding to the range of takeoff angles from 35° to 45° is 0.839 sr, and the solid angle corresponding to the range of takeoff angles from 40° to 50° is 0.774 sr. ***Notes***: **1)** The input files contain the option "tallies_per_source 1", which ensures that the photon numbers in the computed tally data are already given per source electron. **2)** In the case of normal incidence, two photon tallies were calculated: the one mentioned above and another one corresponding to smaller takeoff angles. The name of the photon tally data file corresponding to takeoff angles from 35° to 45° is "PTALLY1_E1.txt".

Each of the mentioned tallies contains three bins. The energy of characteristic X-ray radiation belongs to bin No. 2. Bins No. 1 and No. 3 contain only bremsstrahlung photon counts. Consequently, they may be used to eliminate the contribution of bremsstrahlung to the photon count in bin No. 2. This is achieved by subtracting the average photon count in bins No. 1 and No. 3 from the photon count in bin No. 2.

References

[1] Hughes H. G., *Recent developments in low-energy electron/photon transport for MCNP6* // Progress in Nuclear Science and Technology, Vol. 4 (2014) pp. 454-458

[2] CUDA Toolkit Documentation. [Online]. Available: http://docs.nvidia.com/cuda/index.html

[3] Bote D., Salvat F., Jablonski A. and Powell C. J., *Cross sections for ionization of K, L and M shells of atoms by impact of electrons and positrons with energies up to 1 GeV: Analytical formulas* // At. Data Nucl. Data Tables, vol. 95 (2009) pp. 871-909

[4] Salvat F., Jablonski A. and Powell C. J., *ELSEPA—Dirac partial-wave calculation of elastic scattering of electrons and positrons by atoms, positive ions and molecules* // Comp. Phys. Comm., vol. 165 (2005) pp. 157-190

[5] Trkov A., Herman M. and Brown D. A. (ed.), *ENDF-6 Formats Manual*, National Nuclear Data Center, Brookhaven National Laboratory, Upton, NY 11973-5000, 2012. 378 p.

[6] Köhn C. and Ebert U., *Angular distribution of Bremsstrahlung photons and of positrons for calculations of terrestrial gamma-ray flashes and positron beams* // Atmospheric Research, vol. 135-136 (2014) pp. 432-465

[7] Kim L., Pratt R. H., Seltzer S. M. and Berger R. H., *Ratio of positron to electron bremsstrahlung energy loss: An approximate scaling law* // Phys. Rev. A, vol. 33 (1986) pp. 3002-3009

[8] Slater J. C., *Atomic shielding constants* // Phys. Rev., vol. 36 (1930) pp. 57-64

[9] Frenkel J., *Some remarks on the theory of the photoelectric effect* // Phys. Rev., vol. 38 (1931) pp. 309-320

[10] Salvat F. and Fernandez-Varea J. M., *Overview of physical interaction models for photon and electron transport used in Monte Carlo codes* // Metrologia, vol. 46 (2009) pp. S112-S138

[11] Seltzer S. M., *An Overview of ETRAN Monte Carlo Methods* // Monte Carlo Transport of Electrons and Photons, ed. T. M. Jenkins, W. R. Nelson and A. Rindi, Plenum Press, New York (1988). ISBN 0–306–43099–1

[12] Tsai Y., *Pair production and bremsstrahlung of charged leptons* // Rev. Mod. Phys., vol. 46 (1974) pp. 815-851

[13] Sood A., *Doppler energy broadening for incoherent scattering in MCNP5, Part I*, Los Alamos National Laboratory, LA-UR-04-0487 (2004)

[14] Sood A., *Doppler energy broadening for incoherent scattering in MCNP5, Part II*, Los Alamos National Laboratory, LA-UR-04-0488 (2004)

[15] Biggs F., Mendelsohn L. B. and Mann J. B., *Hartree-Fock Compton profiles for the elements* // Atomic Data and Nuclear Data Tables, Vol. 16, No. 3 (1975), pp. 201-309